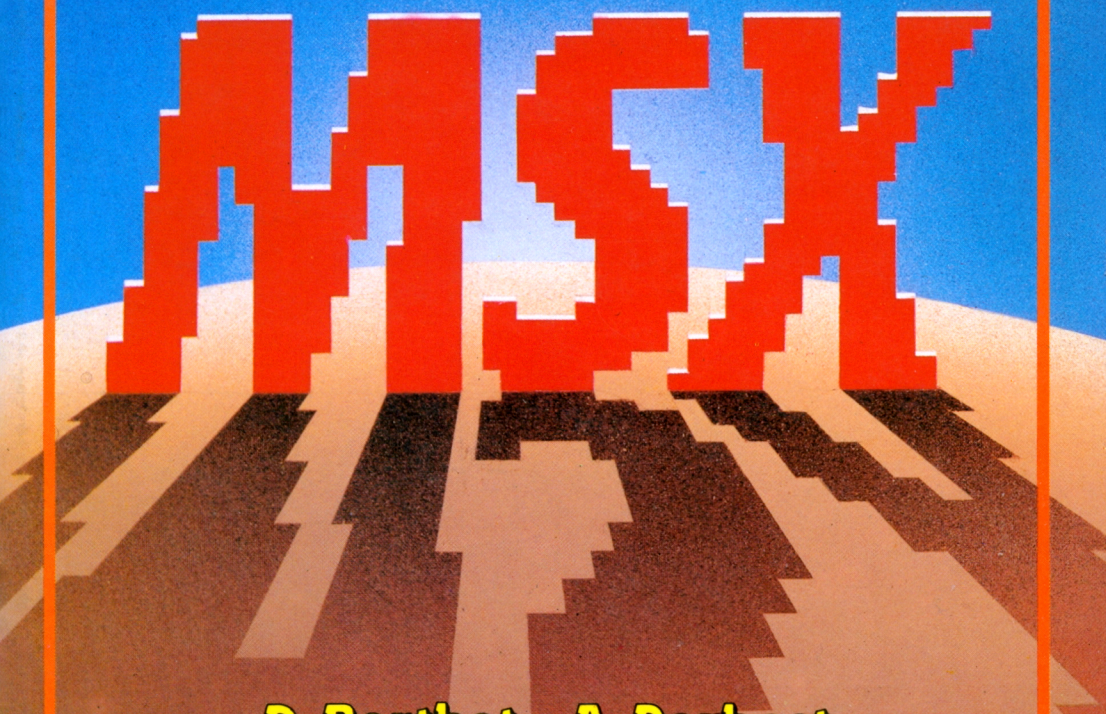
INTRODUCTION A MSX



D. Berthet - A. Perbost

Edimicro

O_ TH. DUVAL 85

INTRODUCTION À MSX

MSX™ est une marque déposée de Microsoft

Couverture : Thierry DUVAL

© F.D.S./Edimicro 1985
Première édition

Imprimé en France. Droits mondiaux réservés.

« La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les "copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective" et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, "toute représentation ou reproduction intégrale ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite" (alinéa 1er de l'article 40). »
« Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal. »

ISBN : 2-904457-29-1

EDIMICRO

DÉPARTEMENT ÉDITIONS DE F.D.S. SARL

121-127, avenue d'Italie-75013 Paris

INTRODUCTION À MSX

par
Alain PERBOST
et
Didier BERTHET

Collection « langages »

Edimicro

Table des matières

INTRODUCTION 11

CHAPITRE 1 : **Programmation et langage Basic** **15**

1.1 Généralités 15
1.2 Démarrage du Basic 17
1.3 Mode calculateur, mode programme 18
1.3.1 Mode calculateur 18
1.3.2 Mode programme 19

1.4 L'éditeur « pleine page » MSX 21
1.4.1 Présentation de l'éditeur MSX 21
1.4.2 Les commandes de l'éditeur 22
1.4.3 Touches de fonctions 24

1.5 Constantes et variables 25
1.5.1 Généralités 25
1.5.2 Constantes et variables numériques entières 27
1.5.3 Constantes et variables numériques réelles simple précision 27
1.5.4 Constantes et variables numériques réelles double précision 28
1.5.5 Notation scientifique 29
1.5.6 Notations hexadécimale, octale, binaire 29
1.5.7 Constantes et variables alphanumériques 31
1.5.8 Tableaux récapitulatifs 32
1.5.9 Déclaration des types de variables 32
1.5.10 Conversion de types 33

1.6 Les opérateurs 35
1.6.1 Opérateurs numériques 35
1.6.2 Opérateurs sur chaînes de caractères 36
1.6.3 Opérateurs relationnels 37
1.6.4 Opérateurs logiques 38

CHAPITRE 2 : **Instructions et fonctions du MSX-Basic** **41**

2.1 Les instructions du MSX-Basic 42
2.1.1 Commandes système 43
2.1.2 Affectations, déclarations 47
2.1.3 Instructions d'usage général 53
2.1.4 Instructions d'entrée-sortie 55
2.1.5 Boucles et branchements 65
2.1.6 Les interruptions 71
2.1.7 Traitement des erreurs 74
2.1.8 Manipulations de fichiers 77

2.2 Les fonctions du MSX-Basic 81
2.2.1 Fonctions d'usage général 82
2.2.2 Calculs 86
2.2.3 Traitement de chaînes de caractères 87

2.3 Contrôle de l'écran 89
2.4 Magnétophone 92
2.5 Manettes de jeux 95
2.6 Le clavier 97

CHAPITRE 3 : **Commandes graphiques et sonores** **101**

3.1 Commandes graphiques 102
3.1.1 Organisation de la mémoire vidéo en haute résolution 102
3.1.2 Instructions de contrôle de l'écran 103
3.1.3 Instructions de visualisation 106
3.1.4 Instructions de tracé 108
3.1.5 Génération et manipulation de lutins 112

3.2 Possibilités sonores du MSX-Basic 119
3.2.1 Génération de notes et de mélodies 120
3.2.2 Génération de sons 123
3.2.3 La fonction PLAY 128

CHAPITRE 4 : **Conclusion** **129**

ANNEXES : 1 Jeu de caractères — caractères spéciaux 131
2 Codes et messages d'erreur 132
3 Sélection des couleurs 135
4 Macro-commandes graphiques 136
5 Macro-langage musique 137
6 Organisation de la mémoire de base 138
7 Spécifications matérielles minimales du système MSX 139

INDEX 140

INTRODUCTION

Les progrès de l'informatique et le nombre croissant de ses applications ont contribué à impliquer le grand public dans le phénomène informatique. L'informatique est maintenant présente dans notre vie quotidienne. Une révolution technologique est ainsi à l'origine d'une véritable révolution de société.

Le marché mondial de la micro-informatique est devenu, en raison de son importance, un enjeu économique. De très grosses sociétés, américaines et japonaises pour la plupart, se livrent une guerre féroce pour conquérir ce marché.

Tous les jours, de nouveaux ordinateurs apparaissent. Seuls quelques initiés arrivent encore à ne pas se perdre entre les différents microprocesseurs (6502, 6809, Z80, 8088, 68000...), les multiples langages de programmation (Fortran, Basic, Pascal, C...), les nombreux systèmes d'exploitation (CP/M, MS-DOS, UNIX...).

L'extrême diversité des matériels et logiciels présents sur le marché de l'informatique est la preuve de la vitalité de ce marché. Malheureusement cette même diversité est aussi la source de bien des déboires pour les pauvres utilisateurs, pris au piège de l'incompatibilité entre les différents systèmes. Ainsi un programme développé sur un Apple II a peu de chances de fonctionner correctement sur un IBM PC : tout le travail est à refaire... De même un périphérique destiné à une machine est généralement inutilisable avec un autre ordinateur.

L'implémentation du langage de programmation Basic sur la plupart des micro-ordinateurs a permis de penser qu'il y avait enfin un point commun entre les différents systèmes. Très vite cependant, de nombreux dialectes du Basic sont apparus, chaque constructeur voulant exploiter au mieux les possibilités matérielles de sa machine (en particulier au niveau du graphisme et des sons).

Or l'incompatibilité matérielle et logicielle des systèmes ne porte pas seulement préjudice aux utilisateurs mais aussi aux contructeurs. La prolifération des différents micro-ordinateurs sature le marché et les fabricants sont obligés de diminuer leur production. L'amortissement des coûts de développement et de fabrication devient donc de plus en plus difficile sur de petites séries. De plus, les producteurs de logiciels hésitent désormais à développer des gros programmes qui ne seront pas « portables » d'un système à l'autre. La diffusion d'un logiciel nécessite en effet l'écriture d'une version spécifique pour chaque machine, ce qui augmente considérablement les coûts de développement.

Consciente de ces problèmes, une grande société américaine, Microsoft (numéro un mondial du Basic), a décidé dans le courant de l'année 1983 d'établir, en accord avec la plupart des grandes firmes japonaises de micro-informatique, les spécifications matérielles et logicielles d'un ordinateur personnel : ainsi naquit le standard MSX (Microsoft Super eXtended). Cette norme a pour but d'unifier enfin le marché micro-informatique, en proposant des micro-ordinateurs performants, compatibles entre eux et de prix raisonnable.

Le cahier des charges très précis de la norme garantit la compatibilité totale entre toutes les machines MSX : les périphériques et les programmes doivent pouvoir s'échanger sans aucune contrainte entre les différents micro-ordinateurs répondant à la norme MSX. Les constructeurs restent bien entendu libres d'ajouter toutes les caractéristiques qu'ils désirent à leurs machines, dans la mesure où ils respectent l'intégrité du noyau MSX, garant de la compatibilité entre les différents micro-ordinateurs MSX.

La norme MSX assure à la fois la compatibilité matérielle et la compatibilité logicielle.

— La compatibilité matérielle est due à l'articulation interne des machines autour de trois circuits principaux :

- un microprocesseur Z80 de Zilog
- un contrôleur de visualisation TMS 9918A ou TMS 9928A de Texas Instuments.
- un générateur de sons AY 3-8910, trois voies, huit octaves.

Vous trouverez en annexe une liste plus complète des spécifications matérielles minimales du standard MSX.

— La compatibilité logicielle provient de l'adoption d'un système d'exploitation appelé MSX-DOS. Ce système d'exploitation est très proche de la version 1.1 du système MS-DOS de la société Microsoft.

Le langage de programmation choisi est une version améliorée du MBasic (4.5) de Microsoft dont il reprend toutes les instructions. Un soin particulier a été apporté aux commandes graphiques et sonores.

Introduction à MSX n'a pas pour but d'être un ouvrage de référence sur toute la norme MSX : notre étude sera plus particulièrement axée sur le MSX-Basic dont nous allons passer en revue toutes les commandes et instructions. Les lecteurs peu familiers du Basic auront ainsi l'occasion de découvrir ce langage de programmation. Les lecteurs déjà initiés trouveront de nombreux détails leur permettant de tirer le meilleur parti des ressources de leurs machines MSX.

Le premier chapitre procède à l'étude de l'éditeur MSX et décrit les opérandes et opérateurs manipulés par le MSX-Basic.

Le second chapitre passe en revue de façon détaillée les instructions et les fonctions du MSX-Basic.

Les instructions graphiques et sonores sont regroupées dans le troisième chapitre.

CHAPITRE 1

Programmation et langage Basic

1.1 GÉNÉRALITÉS

*** Pourquoi programmer un ordinateur ?**

Par lui-même, un ordinateur n'est qu'un fatras de circuits intégrés. Seule l'action de l'homme peut permettre à cet assemblage de réaliser une fonction donnée. Les seules qualités intrinsèques d'un ordinateur sont rapidité, précision, patience infinie et honnêteté sans faille. L'intelligence et l'imagination resteront toujours le propre de l'homme qui programme la machine, celle-ci ayant pour seul mérite d'exécuter scrupuleusement les ordres reçus.

*** Comment peut-on programmer un ordinateur ?**

Un ordinateur ne raisonne qu'en termes de courant (ou de tension) ; le courant passe (niveau haut) : 1, ou ne passe pas (niveau bas) : 0. Il existe donc un langage propre à la machine (que

l'on appelle « langage machine ») qui permet de donner, par une suite de 1 et de 0, des ordres exécutables par l'ordinateur. Cette façon de procéder s'avère généralement longue et complexe.

Des langages de programmation dits « évolués » ont été développés afin de permettre une programmation simple et rapide. Des interpréteurs ou des compilateurs traduisent ensuite les programmes, écrits en langage évolués, en langage machine exécutable par l'ordinateur.

*** Qu'est-ce donc qu'un programme ?**

Un programme est une suite d'instructions que l'utilisateur souhaite voir exécuter dans un ordre déterminé pour réaliser une certaine fonction : jouer aux échecs, faire de la musique, calculer des intégrales...

*** Le langage Basic**

Toutes les machines répondant à la norme MSX sont dotées d'un interpréteur BASIC *(Beginners All-purpose Symbolic Instruction Code)*. Mis au point au cours des années 60, le langage de programmation Basic est devenu de plus en plus puissant et s'est répandu dans le monde entier. Le MSX-Basic est l'une des plus puissantes versions actuelles du Basic. Il est dû à la très grande société américaine Microsoft, déjà connue comme numéro un mondial du Basic avec le MBasic. Les trois lettres MSX signifient Microsoft Super eXtended.

*** Instructions et fonctions**

Comme tout langage, le Basic possède un vocabulaire, une syntaxe et des règles de grammaire. Le vocabulaire est constitué par un ensemble de mots clés tels que PRINT, INPUT... Chacun de ces mots clés représente soit une instruction, soit une fonction.

Une instruction est un ordre qui implique l'exécution d'une tâche précise par l'ordinateur. Certaines instructions peuvent être suivies par un ou plusieurs arguments.

Exemple

PRINT « COUCOU »

PRINT : instruction d'affichage

« COUCOU » : argument de la fonction PRINT ; message à afficher.

Une fonction correspond à une question posée à l'ordinateur qui renvoie alors une réponse. Les fonctions sont généralement utilisées à la suite d'une instruction.

Exemple

LET A = INT (1984/4)
LET : instruction d'affectation
INT : fonction partie entière dont 1984/4 est, dans ce cas, l'argument.

LET A = INT (1984/4) aura pour effet de stocker la valeur 496 dans la variable A.

1.2 DÉMARRAGE DU BASIC

Les spécifications matérielles minimales de la norme MSX prévoient que le système dispose d'une mémoire morte (ROM) de 32 Ko contenant le Basic.

Sous réserve qu'une cartouche d'extension n'ait pas été enfichée dans le port prévu à cet effet, l'ordinateur MSX affiche sur l'écran du téléviseur, dès sa mise sous tension, le message suivant :

MSX BASIC Version 1.Ø
Copyright 1983 by Microsoft
nnnnn Bytes free
OK

La valeur de *nnnnn* dépend de la taille mémoire disponible sur la machine. Cette mémoire vive (RAM) peut aller de 8 Ko (minimum autorisé par la norme MSX) à 64 Ko (espace mémoire maximal adressable par un microprocesseur 8 bits comme le Z80).

L'apparition du symbole *OK* et du curseur signifie que l'interpréteur Basic est prêt à fonctionner. Dès lors, le micro-ordinateur est prêt à servir en mode calculateur ou en mode programme.

1.3 MODE CALCULATEUR — MODE PROGRAMME

1.3.1 Mode calculateur

Le mode calculateur, appelé aussi mode immédiat ou mode commande, ressemble beaucoup au mode de fonctionnement d'une calculatrice. Ce mode permet en effet d'obtenir l'exécution immédiate d'un calcul, d'une instruction ou d'une commande.

Supposons, par exemple, que vous tapiez sur le clavier de votre micro-ordinateur la commande : PRINT «BONJOUR» RC

RC représente la pression de la touche de validation que tout ordinateur possède. Cette touche est parfois appelée RC, CR, NEWLINE, ENTER... Son emploi est obligatoire pour signifier à l'ordinateur que les caractères tapés auparavant représentent une commande à exécuter.

A chaque fois que vous avez tapé l'un des caractères de la commande PRINT «BONJOUR», le micro-ordinateur a rangé ce caractère en mémoire tout en le faisant apparaître sur l'écran du téléviseur.

La pression de la touche RC donne l'ordre à l'interpréteur Basic de se mettre au travail pour décoder et exécuter la commande reçue. Extrayant de la mémoire, les uns après les autres, les caractères entrés précédemment, l'interpréteur a reconnu l'instruction PRINT qui lui ordonne d'afficher un message sur l'écran du téléviseur (PRINT signifie *imprimer* en anglais). Il a ensuite reconnu entre guillemets le message à afficher : «BONJOUR»

Ayant ainsi totalement décodé l'ordre reçu, l'interpréteur Basic a pu passer à la phase d'exécution, d'où l'apparition du message «BONJOUR» sur l'écran.

La réapparition du symbole OK signifie que l'interpréteur Basic est à nouveau prêt pour recevoir et exécuter toute commande.

L'ordinateur peut servir, en mode immédiat, de calculatrice.

Par exemple : PRINT 3 + 2 RC provoque l'apparition du résultat de l'addition, à savoir 5, sur l'écran.

PRINT ((3 + 2) * 5 ↑2)/7 − 1 RC
donne : 16.857142857143

Vous pouvez ainsi calculer la valeur d'expressions arithmétiques et logiques très complexes. La quasi-totalité des instructions et fonctions du MSX-Basic est accessible en mode calculateur.

1.3.2 Mode programme

Le mode calculateur ne permet pas d'exécuter automatiquement une suite d'instructions. Pour ce faire, il est nécessaire d'écrire un programme et donc d'utiliser le mode programme de l'ordinateur. Ce mode permet, comme son nom l'indique, de mettre en mémoire des programmes écrits en Basic puis d'obtenir l'exécution de ces programmes.

Un programme Basic est constitué par une suite de lignes numérotées contenant une ou plusieurs instructions. Dans le cas où une ligne de programme comporte plusieurs instructions, ces instructions doivent être séparées par le caractère « : »

En MSX-Basic, les numéros de ligne sont obligatoirement des nombres entiers compris entre 0 et 65529. Les lignes de programme sont stockées en mémoire et exécutées par ordre croissant de leurs numéros. Une ligne de programme peut contenir au maximum 255 caractères. Cela peut correspondre à plusieurs lignes physiques sur l'écran du téléviseur.

Toute suite d'instructions précédée d'un numéro est considérée par l'interpréteur Basic comme une ligne de programme. La pression de la touche (RC) pour valider la ligne ne provoquera donc pas, à la différence du mode immédiat, l'exécution des instructions mais leur mémorisation en vue d'un usage ultérieur. C'est pourquoi le mode programme est aussi appelé mode différé.

L'exécution du programme stocké en mémoire peut être obtenue avec l'instruction RUN. Des instructions spécialisées telles que LIST, DELETE permettent de travailler sur le programme mémorisé (voir le paragraphe sur les commandes système).

La norme MSX prévoit aussi l'utilisation d'un éditeur de texte « pleine page » dont les caractéristiques seront étudiées dans le paragraphe suivant.

Remarques

* Pour améliorer la lisibilité et faciliter la mise au point des programmes, il est vivement conseillé de numéroter les lignes de 10

en 10. L'emploi de l'instruction RENUM permet d'obtenir très simplement une telle numérotation (voir paragraphe sur les commandes système).

* Les espaces séparant les mots clés des arguments et les différentes instructions ne sont pas significatifs. Il est tout de même conseillé, dans un but de clarté, de conserver ces espaces.

Exemple de programme

```
1Ø CLS
2Ø INPUT «QUEL^EST^VOTRE^PRENOM»; P$
3Ø PRINT
4Ø PRINT «BONJOUR^»; P$
5Ø END
```

^ symbolise l'espace

Si vous tapez RUN et (RC) alors que le petit programme ci-dessus est en mémoire, vous verrez l'écran du téléviseur s'effacer ; l'ordinateur vous demandera alors poliment votre prénom, puis vous dira *bonjour*.

En fin d'exécution le symbole OK réapparaît. Le micro-ordinateur est de nouveau prêt à fonctionner, soit en mode calculateur, soit en mode programme.

Si une erreur se produit en cours d'exécution d'un programme, cette exécution est interrompue et le message d'erreur correspondant est affiché. Vous trouverez en annexe la liste complète des messages d'erreur du MSX-Basic.

Vous pouvez être amené à devoir interrompre volontairement l'exécution d'un programme : boucle de durée infinie... Pour ce faire, il suffit de presser la touche STOP. L'exécution s'arrête et le curseur réapparaît. Sous réserve que vous n'ayez pas fait ensuite de modifications du programme, vous pouvez relancer l'exécution à partir du point d'interruption en pressant simplement la touche STOP, à nouveau.

L'interruption de l'exécution d'un programme par la pression simultanée des touches CTRL et STOP est beaucoup plus sévère. En effet le message *Break in nnnn* apparaît, indiquant le numéro de la ligne qui était en cours d'exécution. Le symbole OK réapparaît lui aussi. La relance du programme nécessite alors l'emploi de l'instruction RUN ou de l'instruction CONT.

1.4 L'EDITEUR « PLEINE PAGE » MSX

1.4.1 Présentation de l'éditeur MSX

L'éditeur de texte MSX est conçu pour être le plus simple d'emploi, le plus efficace et le plus performant possible. Contrairement à la solution malheureusement retenue par la plupart des systèmes d'exploitation équipant les micro-ordinateurs actuels, la norme MSX a en effet prévu l'adoption d'un éditeur de texte « pleine page » (on dit aussi parfois « plein écran »).

Grâce à des touches spéciales, l'utilisateur peut très simplement et très rapidement positionner le curseur en n'importe quel point de l'écran. Rendu au point désiré, l'utilisateur peut alors effectuer des insertions de texte, des modifications ou des corrections avant de valider les changements en pressant la touche RC.

Le texte visualisé sur l'écran (simple texte ou ligne de programme) peut être modifié par :

— réécriture sur des caractères existants
— insertion de caractères
— ajout de nouveaux caractères à la fin d'une ligne logique
— effacement de caractères à gauche du curseur
— effacement de caractères à droite du curseur.

Toutes ces interventions sont possibles en utilisant les commandes de l'éditeur MSX décrites dans le paragraphe 1.4.2.

Les modifications sont validées dès la frappe de la touche RC et concernent toute la ligne logique, quelque soit la position du curseur dans cette ligne au moment de la validation.

En mode direct, c'est-à-dire traitement d'une ligne de texte non précédée par un numéro, l'éditeur MSX qui était actif depuis l'apparition du symbole OK, rend le contrôle à l'interpréteur Basic dès la frappe de la touche RC pour que la commande entrée soit exécutée.

En mode programme, l'éditeur MSX garde le contrôle jusqu'à ce que l'utilisateur donne l'instruction RUN, auquel cas l'interpréteur Basic commencera l'exécution du programme.

Après la frappe de ®, les lignes de programme sont traitées par l'éditeur MSX de la façon suivante :

— Une nouvelle ligne de programme est mémorisée.

Ce cas se produit lorsque la ligne de texte est précédée par un numéro de ligne correct (nombre entier compris entre 0 et 65529) et qu'au moins un caractère autre que l'espace suit le numéro de ligne.

— Une ancienne ligne est modifiée.

Ce cas se produit lorsque le numéro de la ligne logique en cours de traitement correspond au numéro d'une ligne de programme déjà mémorisée. Dans ce cas la nouvelle ligne vient prendre la place de l'ancienne en mémoire.

— Une ligne de programme mémorisée est détruite.

Cela se produit lorsque la ligne logique validée est constituée par un numéro de ligne seulement, correspondant à une ligne déjà stockée.

— Une erreur est produite.

Ce cas peut provenir de la tentative de destruction d'une ligne non existante ou d'un phénomène de saturation mémoire (la mémoire est pleine et la nouvelle ligne ne peut pas être sauvegardée).

1.4.2 Les commandes de l'éditeur MSX

Pour insérer, détruire ou modifier des caractères, l'éditeur MSX dispose de commandes accessibles soit à partir de touches spéciales soit en pressant simultanément la touche CTRL et une autre touche.

Les commandes de l'éditeur ont toutes un code (code inférieur à 32). Ces codes peuvent être utilisés dans un programme Basic en utilisant la fonction CHR$, mais l'effet n'en est pas toujours très intéressant.

Code	Touche spéciale	Touches associées	fonction
1		CTRL A	Néant
2 *		CTRL B	Place le curseur au début du mot précédent.
3 *		CTRL C	Interrompt l'exécution d'un programme en attente d'entrée (instruction INPUT). En édition, re-

				tourne au mode direct sans sauvegarde des modifications en cours.
4	*		CTRL D	Néant.
5	*		CTRL E	Efface tous les caractères situés à droite du curseur jusqu'à la fin de la ligne logique.
6	*		CTRL F	Place le curseur au début du mot suivant.
7	*		CTRL G	Bip sonore
8		Back Space	CTRL H	Détruit le caractère situé à gauche du curseur. Tous les caractères situés à droite du curseur dans la ligne logique sont décalés d'un cran vers la gauche.
9		Tab	CTRL I	Déplace le curseur vers la droite, à la position suivante de tabulation (une position toutes les 8 colonnes).
10	*		CTRL J	Saut d'une ligne.
11	*	Home	CTRL K	Ramène le curseur dans le coin supérieur gauche de l'écran.
12	*	CLS	CTRL L	Efface l'écran et ramène le curseur dans le coin supérieur gauche de l'écran.
13	*	Return	CTRL M	Equivalent à la touche RC.
14	*		CTRL N	Positionne le curseur à la fin de la ligne logique.
15	*		CTRL O	Néant
16	*		CTRL P	Néant
17	*		CTRL Q	Néant
18	*	INS	CTRL R	Passage du mode correcteur au mode insertion et réciproquement. En mode insertion, le curseur est diminué de moitié ; les caractères sont rentrés à droite du curseur. Les anciens caractères à droite se déplacent au rythme de l'insertion. En mode correction, le curseur garde sa taille normale ; l'écriture s'effectue en remplaçant les caractères existants par les nouveaux caractères.
19	*		CTRL S	Néant

20	*		CTRL T	Néant
21	*		CTRL U	Efface toute la ligne logique
22	*		CTRL V	Néant
23	*		CTRL W	Néant
24	*	SELECT	CTRL X	Néant
25	*		CTRL Y	Néant
26	*		CTRL Z	Néant
27		ESC	CTRL [	Néant
28	*		CTRL \	Déplacement du curseur d'un cran vers la droite.
29	*		CTRL]	Déplacement du curseur d'un cran vers la gauche.
30	*		CTRL ↑	Déplacement du curseur d'un cran vers le haut.
31	*		CTRL _	Déplacement du curseur d'un cran vers le bas.
128		DEL	DEL	Détruit le caractère pointé par le curseur.

Remarque

Les fonctions marquées d'un astérisque provoquent le passage automatique du mode insertion au mode correction.

1.4.3 Touches de fonctions

La norme MSX prévoit la disposition sur le clavier de dix touches prédéfinies. Leur fonction est visualisée sur la 24° ligne de l'écran, mais l'utilisateur peut faire disparaître cet affichage pour utiliser la ligne ainsi libérée.

A l'initialisation, les dix fonctions sont :

F1 . . . . . COLOR[b]
F2 . . . . . AUTO [b]
F3 . . . . . GOTO [b]
F4 . . . . . LIST [b]
F5 . . . . . RUN [RC]
F6 . . . . . COLOR 15, 4, 7 [RC]
F7 . . . . . CLOAD «
F8 . . . . . CONT [RC]
F9 . . . . . LIST. [RC] [u] [u]
F10 . . . . . [cls] RUN [RC]

[b] espace
[RC] retour chariot
[u] curseur vers le haut
[cls] effacement de l'écran

Le programmeur dispose d'instructions spéciales pour assigner à ces touches de nouvelles fonctions et pour lire ces touches. Vous trouverez des détails plus complets dans le paragraphe concernant la gestion du clavier.

1.5 CONSTANTES ET VARIABLES

1.5.1 Généralités

Tout programme Basic manipule des données ou opérandes qui sont les valeurs, nombres ou suites de caractères servant d'argument aux instructions et fonctions. Ces opérandes peuvent se présenter de deux façons :

— constantes
— variables

*** Constantes**

Le MSX-Basic permet d'utiliser des constantes numériques et des constantes « chaînes » (on dit aussi constantes alphanumériques). Les constantes numériques peuvent être de type entier, réel (simple ou double précision). Une constante « chaîne » est une suite de caractères encadrée par des guillemets. Une chaîne peut contenir au maximum 255 caractères autres que des guillemets.

*** Variables**

Une variable est un emplacement particulier de la mémoire auquel le programmeur peut donner le nom qu'il désire. Cet emplacement peut voir son contenu « varier », d'où le nom de variable. Les variables peuvent avoir, comme les constantes, différents types : entier, réel simple ou double précision, chaîne de caractères.

Le MSX-Basic permet des noms de variables ayant une longueur quelconque, mais seuls les deux premiers caractères sont significa-

tifs. Les noms de variables peuvent comporter librement des lettres et des chiffres, mais le premier caractère doit obligatoirement être une lettre. Un nom de variable ne doit pas être un mot-clé du langage Basic et ne peut pas commencer par un mot réservé du langage. Ainsi le nom TOTO est interdit car il commence par TO qui fait partie de la structure FOR-TO-NEXT. Le nom d'une variable alphanumérique doit être suivi du symbole $.

Exemples

TAB1 TAB2	ces noms représentent la même variable.
1A	interdit car le premier caractère est un chiffre
FORME	interdit car commence par FOR.
A1B2C3	autorisé
SCORE$	autorisé

Avant affectation d'une valeur par le programme, toute variable numérique est mise à zéro et toute variable alphanumérique est initialisée à « » (chaîne vide).

*** Tableaux**

Un tableau est un ensemble de variables de même type, référencé par un nom unique. Chaque élément du tableau est représenté par le nom du tableau suivi d'un ou plusieurs indices (il faut autant d'indices qu'il y a de dimensions au tableau).

Exemples

A(2) est un élément d'un tableau à une seule dimension.

T(3,5) est un élément d'un tableau à deux dimensions.

Le MSX-Basic autorise des tableaux ayant jusqu'à 255 dimensions, chacun des indices pouvant prendre des valeurs de 0 à 32767. Les noms des tableaux suivent les mêmes règles que pour les variables numériques et alphanumériques.

Exemples

TAB1(1)
TAB2(1) représentent le même élément

AB$(5) représente le sixième élément d'un tableau de chaînes de caractères.

Remarques

* Les indices d'un tableau peuvent être des constantes ou des variables. Un indice doit toujours être un nombre entier positif ou nul. Si on désigne donc un élément de tableau par des indices :

— non entiers, l'ordinateur les transforme en les indices entiers les plus proches ;
— négatifs, une erreur est produite.

* Le premier indice utilisable est Ø.

* Le nombre maximal d'éléments pouvant être stockés dans un tableau dépend de la taille mémoire disponible sur le micro-ordinateur.

* Pour des tableaux dont les indices ne dépassent pas 1Ø, il n'est pas nécessaire d'utiliser l'instruction de déclaration DIM (voir paragraphe sur les instructions d'affectation et de déclaration). Cela est quand même vivement conseillé.

1.5.2 Constantes et variables numériques entières

Les constantes et variables numériques entières peuvent prendre toutes les valeurs entières comprises entre – 32768 et 32767 (sotckage sur 2 octets en utilisant la convention du complément à 2).

Le type entier d'une constante ou d'une variable numérique est indiqué par le suffixe %.

Exemples

1985% constante numérique entière
A% variable numérique entière

Si le programmeur désire travailler avec des nombres entiers plus grands que 32767 en valeur absolue, il doit employer des constantes ou variables numériques réelles.

1.5.3 Constantes et variables numériques réelles simple précision

Le nombre de chiffres significatifs utilisés pour les calculs est caractéristique de la précision d'une constante ou d'une variable numérique réelle.

En simple précision, les chiffres significatifs sont au nombre de six. Les constantes et variables numériques réelles simple précision peuvent prendre des valeurs allant en valeur absolue de 1.10^{-63} à 1.10^{+63} (codage sur 4 octets : 3 octets de mantisse plus un octet pour l'exposant).

Le suffixe indiquant qu'une variable ou une constante numérique est de type réel simple précision est !

Exemples

3.33 ! constante réelle simple précision
A ! variable réelle simple précision

1.5.4 Constantes et variables numériques réelles double précision

En double précision le nombre de chiffres significatifs est 14. Les constantes et variables numériques réelles double précision peuvent prendre des valeurs allant en valeur absolue de 1.10^{-63} à 1.10^{+63} (codage sur 8 octets : 7 octets de mantisse et un octet pour l'exposant).

Le type réel double précision est indiqué par le suffixe #. La présence de ce suffixe n'étant pas obligatoire, on en déduit que, par défaut de suffixe, les variables et constantes numériques seront de type réel double précision.

Exemples

– 1.495# constante réelle double précision
A# variable réelle double précision
B " " " "

Considérons le petit programme ci-dessous pour récapituler les différents cas exposés.

```
1Ø CLS
2Ø INPUT «DONNEZ UN NOMBRE»; A#
3Ø PRINT «DOUBLE PRECISION (A#) :»; A#
4Ø A! = A#
5Ø PRINT «SIMPLE PRECISION (A!) :»; A!
6Ø A % = A#
```

```
7Ø PRINT «ENTIER (A %) :»; A %
8Ø END
```

L'exécution de ce programme en répondant 1.23456789 123456789 à la question : DONNER UN NOMBRE donne les résultats suivants :

```
DOUBLE PRECISION (A#) : 1.23456789123456
SIMPLE PRECISION (A!) : 1.23457
ENTIER (A %) : 1
```

Quatorze chiffres sont donc affichés en double précision ; six seulement subsistent en simple précision alors que la variable A % ne contient plus que la partie entière du nombre d'origine.

Remarque

Pour le MSX-Basic des variables de même nom mais de types différents sont différentes. A#, A! et A % sont donc trois variables différentes.

1.5.5 Notation scientifique

Les constantes numériques réelles en simple et double précision peuvent être représentées en notation scientifique (mantisse plus exposant). La mantisse et l'exposant sont séparés par un symbole indiquant si la constante est en simple ou double précision :

E simple précision
D double précision

Exemples

235.34 E − 5 simple précision
− 1.Ø9 D 3 double précision

1.5.6 Notations hexadécimale, octale, binaire

*** Notation hexadécimale**

Les constantes numériques représentées sous forme hexadécimale (base 16) sont précédées par le symbole & H.

Exemples

&H7FF
&H8E

*** Notation octale**

Les constantes numériques représentées sous forme octale (base 8) sont précédées par le symbole &O (le O est optionnel).

Exemples

&0347
&123

*** Notation binaire**

Les constantes numériques représentées sous forme binaire (base 2) sont précédées par le symbole &B.

Exemples

&B1Ø11Ø1
&BØ1Ø11111Ø1

Remarques

* Il faut faire attention à n'employer, en hexadécimal, que les chiffres Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F, en octal les chiffres Ø, 1, 2, 3, 4, 5, 6 et 7, en binaire les chiffres Ø et 1.

* A l'affichage les constantes numériques exprimées sous forme hexadécimale, octale ou binaire sont restituées en base 10.

Exemple

PRINT &B11111111 donne 255 sur l'écran.

*** Le passage de la représentation décimale d'un nombre à ses représentations hexadécimale octale et binaire se fait avec les fonctions HEX$, OCT$ et BIN$**

1.5.7 Constantes et variables alphanumériques

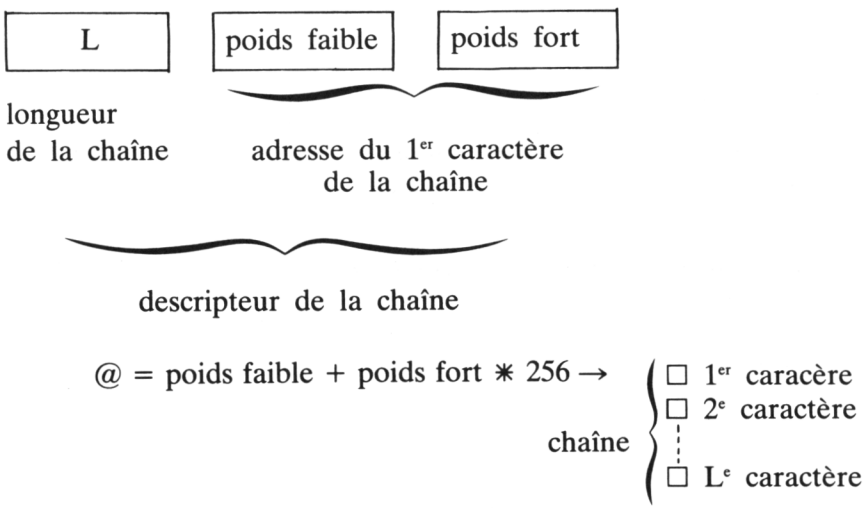
Nous avons déjà vu qu'une constante de type chaîne de caractères est une suite de 255 caractères au maximum encadrée par des guillemets.

Le suffixe indiquant qu'une variable est de type chaîne est le symbole $ accolé au nom de la variable.

Exemples

NOM$ =	« EDIMICRO »
variable alphanumérique	constante alphanumérique
VILLE$(1Ø) =	« PARIS »

Le stockage d'une chaîne de caractères fait appel à un descripteur de variable constitué par 3 octets contenant la longueur de la chaîne et l'adresse de stockage du premier caractère de la chaîne.



La longueur de la chaîne est stockée sur un octet, ce qui explique que l'on ne puisse pas dépasser une longueur de 255 caractères.

Remarque

A l'initialisation du système MSX, une place mémoire de 200 octets seulement est réservée pour le stockage des chaînes. C'est assez peu et il faudra prévoir de réserver éventuellement davantage de place avec l'instruction CLEAR.

1.5.8 Tableaux récapitulatifs

*** Plages de variations**

type	nom de variable	valeurs extrêmes
entier	X %	− 32768 ; + 32767
réel simple précision	X !	$\pm 1.10^{+63}$
réel double précision	X# (ou X)	$\pm 1.10^{-63}$
chaîne de caractères	X$	0 à 255 caractères

*** Encombrement mémoire des variables**

type	nombre d'octets occupés
entier	2
simple précision	4 (mantisse : 3, exposant : 1)
double précision	8 (mantisse : 7, exposant : 1)
chaîne de caractères	3 + nombre de caractères

1.5.9. Déclaration des types des variables

Nous avons vu que l'on pouvait définir le type d'une variable en juxtaposant un suffixe avec son nom :

% variable entière
! variable réelle simple précision
# variable réelle double précision
$ variable chaîne de caractères

Cette méthode est contraignante et peu économe en place mémoire puisque l'on retrouve le suffixe chaque fois que le nom de la variable est utilisé. Pour remédier à cet inconvénient, le MSX-Basic dispose de quatre instructions spéciales qui permettent,

en début de programme, de spécifier quelles sont les variables qui seront entières, réelles en simple ou en double précision, alphanumériques. Il s'agit des instructions DEFINT, DEFSNG, DEFDBL, DEFSTR.

La syntaxe générale de ces instructions est :

DEF < type de variable > [suite de lettres]

Type de variable :

INT entier
SNG réel simple précision
DBL réel double précision
STR chaîne de caractères

Exemples

DEFINT A-H, P-Z force le type de toutes les variables dont le nom commence par une lettre comprise entre A et H ou P et Z au type entier.

DEFSNG B implique que toutes les variables dont le nom commence par B seront de type réel simple précision.

DEFDBL N-T signifie que toutes les variables dont le nom commence par une lettre comprise entre N et T seront de type réel double précision.

DEFSTR B, C, D entraîne que toutes les variables dont le nom commence par B, C ou D seront de type chaîne de caractères.

Remarque

Les suffixes %, !, # et $ sont prioritaires devant les instructions DEFINT, DEFSNG, DEFDBL et DEFSTR. Cela permet de forcer le type d'une variable à un type autre que celui qui avait été déclaré en début de programme.

1.5.10 Conversion de types

*** Conversion automatique**

Il est fréquent que l'on affecte à une variable d'un type donné la valeur d'une variable ou d'une constante d'un autre type. La conversion de type est, dans ce cas, automatique. Trois cas sont étudiés ci-dessous avec des exemples.

• 1er cas : une constante numérique d'un type quelconque est assignée à une variable d'un type différent. La valeur est alors mémorisée sous la forme imposée par le type de la variable. Le seul cas impossible est l'affectation d'une constante numérique à une variable de type chaîne, cas dans lequel il y apparition d'un message d'erreur (type Mismatch).

Exemple

```
10 A % = 33.12
20 PRINT A %
```

L'exécution de ce programme provoque l'affichage de 33.

• 2e cas : lors du calcul d'une expression, tous les opérandes intervenant dans cette expression sont convertis au meilleur degré de précision possible.

Exemples

```
10 D = 5/9!
20 PRINT D
```

L'exécution donne : .55555555555556

```
10 D! = 5/9
20 PRINT D!
```

L'exécution donne : .555556. L'opération est effectuée en double précision (type de la constante 5/9) mais le résultat est mémorisé en simple précision (type de la variable D!).

• 3e cas : les opérateurs logiques convertissent leurs opérandes en entiers et donnent un résultat entier. Les opérandes doivent donc être compris entre − 32 768 et + 32 767 sinon il y a apparition d'un message de dépassement (Overflow).

*** Conversion volontaire à l'aide des fonctions CINT, CSNG, CDBL, STR$ et VAL**

• C/NT (X) convertit X en nombre entier en supprimant la partie décimale. X doit être compris entre − 32768 et + 32767.

• CSNG (X) convertit X en réel simple précision.

• CDBL (X) convertit en réel double précision.

• STR$ (X) convertit une constante ou une expression numérique en une chaîne de caractères. Cette fonction est utilisée pour permettre des opérations réservées d'habitude aux chaînes de caractères sur des nombres. Attention, tout nombre positif est précédé d'un espace (emplacement réservé pour le signe). Cet espace est conservé lors de la conversion du nombre en chaîne.

• VAL (X$) donne l'équivalent numérique d'une chaîne de caractères.

— Si X$ contient un nombre, VAL donne ce nombre.

— Si X$ contient un nombre et des caractères, VAL ne donne ce nombre que s'il est placé au début de la chaîne de caractères.

— Si le contenu de X$ ne peut pas être « évalué », VAL renvoie Ø.

— Les espaces contenus dans X$ ne sont pas significatifs pour la fonction VAL.

— Les suffixes &H, &O et &B sont pris en compte par la fonction VAL.

Exemples

VAL «1234» donne 1234
VAL «121 PARIS» donne 121
VAL «Y = SIN (11/2)» donne Ø
VAL «2*3» donne 2
VAL «&BØØØØ1111» donne 15

1.6 LES OPERATEURS

Il y a quatre types d'opérateurs, à savoir les opérateurs numériques, les opérateurs sur chaînes de caractères, les opérateurs relationnels et les opérateurs logiques.

1.6.1 Les opérateurs numériques

Très connus, ces opérateurs sont rappelés par ordre de priorité décroissante.

( ) parenthèses qui permettent de délimiter des expressions

—	opérateur unaire qui permet de changer le signe d'une constante ou d'une variable. Exemple : − 3
↑	élévation à la puissance
* /	multiplication (*) et division (/)
\ MOD	division entière (\) et modulo (MOD)
+ −	addition (+) et soustraction (−)

Dans le cas de la division entière, les opérandes sont d'abord convertis en entiers, la division est effectuée puis le résultat est converti en entier. De même l'opération MODULO se fait entre opérandes convertis au préalable en entiers et le résultat est en entier. Cela suppose donc que les opérandes et le résultat sont toujours compris entre − 32768 et + 32767. Dans le cas contraire, un message de dépassement *(Overflow)* apparaît.

Exemples

10\4 = 2
37.52\7.99 = 5
10 MOD 4 = 2
37.52 MOD 7.99 = 2

Si, pendant le calcul d'une expression, une division par 0 doit être faite, le calcul est interrompu et le message *Division by zero* apparaît.

1.6.2 Opérateurs sur chaînes de caractères

Un seul opérateur, appelé concaténation, est utilisable avec les chaînes de caractères. Son symbole est +

Exemple

PRINT «BONJOUR␣» + «MONSIEUR»
donne BONJOUR␣MONSIEUR

␣ représente le caractère espace.

Remarque

La chaîne, résultat de la concaténation de deux autres chaînes, doit contenir au maximum 255 caractères pour pouvoir être sauvegardée.

1.6.3 Opérateurs relationnels

Les opérateurs relationnels sont utilisés pour comparer des valeurs entre elles. Souvent associés à des instructions de test telles que IF-THEN, tous les opérateurs relationnels nécessitent deux opérandes, soit numériques, soit alphanumériques.

Le résultat d'une opération relationnelle ne peut prendre que deux valeurs : « vrai » (valeur − 1)
« faux » (valeur Ø).

Les opérateurs relationnels ont tous la même priorité, inférieure à celles des opérateurs numériques. Cela implique que dans la comparaison d'expressions arithmétiques complexes, les expressions seront calculées avant d'être comparées.

Les opérateurs relationnels sont :

=	égal à
< >	différent de
>	strictement supérieur à
<	strictement inférieur à
> =	supérieur ou égal à
< =	inférieur ou égal à

* Du fait que le résultat d'une « comparaison » est une valeur Ø ou − 1, les opérateurs relationnels peuvent être introduits dans des expressions numériques.

Exemple

```
1Ø CLS
2Ø INPUT A, B
3Ø PRINT A = B
4Ø C = 3 * (A = B) + (A< B)
5Ø PRINT «C =»; C
```

L'exécution de ce programme avec :

A = 2
B = 2

donne l'affichage suivant

− 1	*A = B est vrai*
C = − 3	*A < B est faux*

avec	A = 2	
	B = 3	
donne l'affichage suivant	Ø	*A = B est faux*
	C = − 1	*A < B est vrai*

* Les opérateurs relationnels peuvent mettre en jeu des opérandes alphanumériques. Dans ce cas la comparaison s'effectue caractère par caractère en partant du premier élément à gauche de chacune des deux chaînes. Le classement se fait par ordre croissant des codes ASCII qui correspond justement à l'ordre alphabétique pour les lettres.

Exemple

« TOTO » > « TITI »

* Plusieurs opérations relationnelles peuvent être combinées au moyen d'opérateurs logiques.

1.6.4 Les opérateurs logiques

Les opérateurs logiques sont utilisés pour comparer entre elles des valeurs numériques ou des expressions comportant des opérateurs relationnels, en suivant les règles de la logique booléenne.

Le résultat d'une opération logique est soit vrai (1 logique qui correspond à la valeur − 1 en Basic), soit faux (0 logique qui correspond à la valeur Ø). Dans une expression associant plusieurs types d'opérateurs, les opérateurs sont pris en compte après les opérateurs arithmétiques et relationnels.

Les tables de vérité des opérateurs logiques sont données ci-dessous, les opérateurs étant classés par ordre de priorité décroissante.

NOT — négation

X	NOT X
0	1
1	0

AND

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

et logique

OR

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

ou logique

XOR

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

ou exécutif logique

EQU

X	Y	X EQU Y
0	0	1
0	1	0
1	0	0
1	1	1

équivalent logique

IMP

X	Y	X IMP Y
0	0	1
0	1	1
1	0	0
1	1	1

implication logique

Dans ces tables de vérité, le 1 logique peut être remplacé par le – 1 obtenu comme résultat d'une opération relationnelle.

Exemple

```
1Ø A = 2 : B = 4
2Ø C = 3 : D = 7
3Ø E = A < B AND C < D
4Ø PRINT E
```

L'exécution de ce programme donne E = − 1

En effet A < B vaut − 1
C < D vaut − 1

d'où E = − 1 AND − 1
E = − 1

* Dans le cas d'opérations logiques sur des nombres, les opérandes sont d'abord convertis en entiers (donc sur 16 bits en complément à 2). L'opération logique est ensuite réalisée bit à bit sur les deux entiers. Ainsi 4 OR 5 est équivalent à &BØØØØØ1ØØ OR &BØØØØØ1Ø1.

* Une utilisation classique des opérateurs logiques est le test de bit par masquage avec l'opérateur AND ou par association de bits avec l'opérateur OR.

Exemple

• On désire connaître la valeur du quatrième bit d'un octet (test d'un mot d'état d'un périphérique par exemple).

Ø1Ø11ØØ1 AND ØØØØ1ØØØ = ØØØØ1ØØØ
ØØ1ØØØØ1 AND ØØØØ1ØØØ = ØØØØØØØØ

• On désire associer sur le même octet deux informations contenues dans deux bits de deux octets différents.

ØØØØ1ØØØ OR ØØØØØØØØ1 = ØØØØ1Ø Ø1

• On désire trouver le plus grand entier Y inférieur à X et divisible par 4

Y = X AND &B1111111111111111ØØ

• On désire trouver le plus petit entier supérieur à X

Y = X OR &BØØØØØØØØØØØØØØØØ1

CHAPITRE 2

Instructions et fonctions du MSX-Basic

Ce chapitre étudie en détail toutes les instructions et fonctions du MSX-Basic autres que les instructions graphiques et sonores, qui seront l'objet du chapitre suivant. Les instructions concernant le contrôle de l'écran, l'utilisation d'un magnétophone à cassettes et les manettes de jeux seront également étudiées dans ce chapitre.

Conventions de syntaxe

— Tout ce qui est écrit entre parenthèses () doit être présent au moment de l'édition de l'instruction.

— Tout ce qui est écrit entre crochets est optionnel.

— Les termes entre chevrons < > doivent être remplacés par les valeurs appropriées.

Exemple

LIST [<numéro de ligne>] [–[<numéro de ligne>]]

LIST
LIST 1Ø
LIST 2Ø-4Ø
LIST 5Ø-
LIST -1ØØ

instructions syntaxiquement correctes

L'utilisation d'une instruction syntaxiquement incorrecte est détectée par l'ordinateur qui affiche alors le message d'erreur correspondant : *Illegal Function Call* ou *Syntax Error.*

2.1 LES INSTRUCTIONS DU MSX-Basic

Les instructions générales du MSX-Basic peuvent être classées en huit groupes.

— Commandes système

AUTO	NEW
CONT	RENUM
DELETE	RUN
LIST	TRON/TROFF
LLIST	

— Instructions d'affectation et de déclaration

LET	VARPTR
SWAP	PEEK
DIM	POKE
ERASE	
DEFN	
DEFUSR	

— Instructions d'usage général

BEEP	REM
CLEAR	STOP
END	WAIT

— Instructions d'entrée-sortie

DATA	PRINT
READ	PRINT USING
RESTORE	LPRINT
INPUT	LPRINT USING

LINE INPUT TAB
INPUT$ OUT
INKEY$
INP

— Boucles et branchements

FOR/NEXT/STEP GOTO
GOSUB ON GOTO
RETURN IF/THEN/ELSE
ON GOSUB

— Les interruptions

ON INTERVAL INTERVAL
ON STOP STOP

— Le traitement des erreurs

ON ERROR GOTO ERROR
ERR RESUME
ERL

— Manipulation des fichiers

OPEN LINE INPUT#
CLOSE PRINT#
MAXFILES EOF
INPUT#

2.1.1 Les commandes système

Notre étude débute par les commandes système qui regroupent les instructions plus particulièrement utilisées pour l'écriture et la mise au point de programmes. Ces commandes sont généralement employées en mode direct (d'où le nom de *commande* plutôt que d'*instruction*).

*** AUTO [<numéro de ligne>[,<incrément>]]**

Cette commande permet la génération automatique des numéros de ligne à partir de <numéro de ligne> avec un pas de <incrément>.

Par défaut l'incrément et le numéro de ligne sont initialisés à 10 (AUTO est donc équivalent à AUTO 10,10). Si l'indication <numéro

de ligne> est seulement suivie d'une virgule, la génération automatique s'effectue avec un pas égal au dernier incrément utilisé.

Si un numéro de ligne généré automatiquement est identique au numéro d'une ligne déjà existante, un astérisque (*) apparaît à droite de ce numéro. Toute chaîne de caractère entrée à partir de cet instant remplacera l'ancienne ligne. Par contre, si un simple RC est frappé, l'ancienne ligne est conservée intacte.

Pour sortir du mode AUTO, il suffit de taper CTRL/STOP.

*** CONT**

Cette commande permet de reprendre l'exécution d'un programme à l'endroit où celui-ci a été interrompu par la commande STOP ou CTRL/STOP. Cela suppose bien sûr qu'aucune modification n'a été faite sur le programme pendant l'interruption de son exécution.

*** DELETE [<numéro de ligne>][−[[<numéro de ligne>]]]**

Cette commande permet la destruction de lignes de programme.

Si deux numéros de lignes sont spécifiés, la destruction des lignes de programme s'effectue à partir du premier numéro indiqué jusqu'au deuxième numéro inclu. Si un seul numéro est entré, la commande n'agit que sur la ligne correspondante.

Si l'un des numéros de lignes spécifiés n'existe pas, un message d'erreur apparaît.

DELETE ne s'utilise qu'en mode direct.

Exemples

DELETE 3Ø−6Ø	détruit toutes les lignes de 3Ø à 6Ø.
DELETE 6Ø	détruit la ligne 6Ø.
DELETE 5Ø−	supprime toutes les lignes à partir de la ligne 5Ø incluse.
DELETE −1ØØ	détruit toutes les lignes depuis le début du programme jusqu'à la ligne 1ØØ incluse.

*** LIST [<numéro de ligne>[−[<numéro de ligne>]]]**

Cette commande permet l'impression d'une partie ou de la totalité d'un programme.

Si les deux numéros de lignes sont mis, le programme entier est listé sur l'écran. Le listing peut être interrompu temporairement en pressant la touche STOP. Une nouvelle pression de STOP relance le listing à partir du point d'interruption. La pression simultanée de CTRL/STOP interrompt définitivement le listing.

Si les numéros spécifiés ne correspondent pas à des numéros de lignes existantes, les valeurs prises seront les valeurs réelles les plus proches des valeurs spécifiées.

Exemples

LIST	liste tout le programme
LIST 3Ø-6Ø	liste le programme de la ligne 3Ø à la ligne 6Ø
LIST 2Ø	liste uniquement la ligne 2Ø
LIST –1ØØ	liste le programme depuis le début jusqu'à la ligne 1ØØ
LIST 5Ø–	liste tout le programme à partir de la ligne 5Ø.

*** LLIST [<numéro de ligne>[–[<numéro de ligne>]]]**

Cette commande a le même effet que LIST mais l'impression a lieu sur l'imprimante et non pas sur l'écran comme précédemment. Les possibilités des arguments de LLIST sont les mêmes que pour LIST.

*** NEW**

Cette commande efface tout le programme et toutes les variables de la mémoire de travail. Un autre programme peut alors être chargé en mémoire centrale sans crainte d'interférences avec l'ancien travail de l'ordinateur.

*** RENUM [[<nouveau numéro>][,[<ancien numéro>] [,<incrément>]]]**

Cette commande permet la renumérotation des lignes d'un programme. La renumérotation commence à partir de <ancien numéro> qui prend la valeur <nouveau numéro>. Les nouveaux numéros sont séparés par un pas de <incrément>.

Si <nouveau numéro> est omis, la renumérotation commence à 1Ø. Si <ancien numéro> est omis, la renumérotation débute à la première ligne du programme. Si <incrément> est omis, la valeur par défaut est 1Ø.

Exemples

RENUM 1ØØ,1Ø,2Ø
renumérote les lignes à partir de la ligne 1Ø qui deviendra la ligne 1ØØ. Le pas de renumérotation est fixé est 2Ø.

RENUM 1ØØ
renumérote les lignes à partir du début du programme. La nouvelle numérotation commence à 1ØØ avec un pas de 1Ø.

La commande RENUM est particulièrement utile après la mise au point d'un programme pour obtenir un espacement régulier des numéros de lignes.

Remarques

- RENUM change aussi les numéros de ligne qui suivent les instructions GOTO, GOSUB, THEN, ELSE, ON... GOTO, ON... GOSUB, ON ERROR GOTO et ERL. Si une de ces instructions fait référence à un numéro de ligne non existant, le message « *Undefined line nnnn in mmmm* » est affiché. La référence incorrecte *(nnnn)* n'est alors pas modifiée par RENUM mais le numéro *mmmm* est changé.
- RENUM ne peut pas servir à intervertir des lignes de programme ni pour créer des numéros de ligne supérieurs à 65529.

*** RUN <numéro de ligne>]**

Cette commande permet de lancer l'exécution du programme stockée en mémoire centrale.

Si <numéro de ligne> est spécifié, l'exécution débute à partir de <numéro de ligne>, sinon l'exécution commence à partir de la première ligne du programme.

Remarque

RUN efface toutes les variables de la mémoire avant de lancer l'exécution du programme. Si cela présente un inconvénient, il faudra avoir recours à l'instruction GOTO <numéro de ligne>.

*** TRON/TROFF**

La commande TRON permet de passer du mode d'exécution normale au mode trace. Cette commande peut être soit donnée en mode direct, soit programmé.

Le mode trace se caractérise par le fait que le micro-ordinateur affiche à l'écran chaque numéro de ligne exécutée en cours de programme. Ces numéros sont affichés entre crochets []. Ce mode est très utile pour la mise au point des programmes. Il permet en effet de suivre pas à pas l'exécution du programme et de vérifier que l'enchaînement des opérations correspond bien à ce qu'avait prévu le programmeur.

La commande TROFF permet de repasser du mode trace au mode normal.

Remarque

La commande NEW provoque aussi le passage du mode trace au mode normal (dans le cas où l'ordinateur est en mode trace).

2.1.2 Affectations, déclarations

*** [LET] <variable>=<expression>**

L'instruction d'affectation LET permet d'attribuer à une variable une valeur donnée (valeur d'une constante ou résultat de l'évaluation d'une expression contenant des opérateurs arithmétiques, logiques et relationnels).

Exemples

LET A = 1Ø donne à la variable A la valeur 1Ø

LET P$=«BONJOUR» donne à la variable «chaîne» P$ la valeur «BONJOUR».

LET C = (A<B)*(3↑(2−(A+B<Ø))) est correct.

Comme sa syntaxe l'indique, l'instruction LET est facultative. A = 1Ø a exactement le même effet que LET A = 1Ø. Attention, même si LET est omis, il est sous-entendu. En effet I = I+1 signifie que nouvelle valeur de I = ancienne valeur de I+1 et non pas Ø=1 ! Pour éviter cette confusion, d'autres langages de programmation utilisent un symbole spécial pour l'affectation plutôt que l'opérateur relationnel = (par exemple I ← I+1 ou I : = I+1).

*** SWAP <variable>, <variable>**

Cette instruction permet d'échanger les valeurs de deux variables

de même type. Le message d'erreur « *Type Mismatch* » apparaît si les deux variables sont de types différents.

Exemples

```
SWAP A!,B!
SWAP P$,R$
```

L'instruction SWAP est particulièrement utile pour les programmes de tri. Elle évite au programmeur d'avoir recours à des variables intermédiaires pour faire des permutations dans un tableau ou une liste.

*** DIM <tableau 1>],<tableau 2>...]**

avec <tableau>=<nom de tableau>(dim1[,dim2...]...)

DIM a pour but de déclarer les dimensions des tableaux avant leur utilisation.

Cette déclaration est facultative pour des tableaux dont les indices ne dépasseront pas la valeur 1Ø. Si un indice venait par mégarde prendre une valeur supérieure à 1Ø, l'erreur « *Subscript Out of range* » serait produite.

Remarques

- Il est conseillé de toujours déclarer les tableaux utilisés.
- Un tableau peut avoir jusqu'à 255 dimensions. Le nombre total d'éléments stockés est fonction de la taille mémoire disponible.
- Le premier indice utilisable est Ø.
- Tous les types de variables peuvent être organisés en tableaux.
- Toute tentative de redimensionner un tableau déjà dimensionné auparavant provoquera une erreur. Il faut utiliser d'abord soit CLEAR, soit ERASE, soit RUN qui effacent les variables de la mémoire.
- L'allocation de mémoire est statique pour les tableaux de variables numériques. La réservation se fait à la déclaration. Par exemple DIM A %(1Ø) provoquera la réservation de 11×2=22 octets (les entiers sont codés sur 2 octets). A l'initialisation tous les éléments du tableau sont mis à Ø.

• L'allocation de mémoire est dynamique pour les tableaux de chaînes de caractères. Seuls les descripteurs de chaîne (3 octets par descripteur) nécessitent une allocation statique. A l'initialisation toutes les chaînes sont mises à « » (chaîne vide).

Exemples

```
DIM A(2,3,4), P$(2Ø), BB!(1Ø,1Ø)
DIM T %(2,2)
```

*** ERASE <tableau 1>[,<tableau 2>...]**
avec <tableau>=<nom de tableau> (dim1[,dim2...]...)

Cette instruction permet d'effacer un ou plusieurs tableaux de la mémoire sans toucher aux autres variables. Cela permet de récupérer la place mémoire occupée par certains tableaux dès qu'ils sont devenus inutiles. On peut aussi redimensionner un tableau qui vient d'être éliminé par ERASE.

Exemple

```
ERASE A(2,3,4)
```

*** DEFINT, DEFSNG, DEFDBL, DEFSTR**

Voir paragraphe 1.5.9.

*** DEF FN < nom de fonction > [(< liste de para mètres>)]=<définition de la fonction>**

Cette instruction sert au programmeur pour définir des fonctions qui lui sont utiles et qui ne figurent pas dans la bibliothèque de fonctions du MSX-Basic.

<nom de fonction> est un nom qui obéit aux règles régissant les noms de variables (le premier caractère doit être une lettre, seuls les deux premiers caractères sont significatifs...). Ce nom précédé de FN sera le nom d'appel de la fonction dans toute la suite du programme.

La présence de <liste de paramètres> est optionnelle. Cette liste peut être constituée par une suite de noms de variables séparés par une virgule.

<définition de la fonction> est une expression utilisant ou non les variables spécifiées dans <liste de paramètres>.

Lors de l'évaluation de l'expression, les paramètres qui figurent dans <liste de paramètres> sont passés par programme lors de l'appel de la fonction. Ceux qui ne figurent pas dans cette liste se voient affecter les valeurs des variables correspondantes.

Les paramètres qui figurent dans <liste de paramètres> et dans <définition de la fonction> sont des paramètres purement formels et n'affectent en rien les variables du programme qui portent le même nom.

Remarques

- <définition de la fonction> ne peut pas dépasser une ligne de programme. Au-delà il faut recourir aux sous-programmes.
- <nom de fonction> force le type de la fonction. <définition de la fonction> doit avoir le même type sinon une erreur *Type Mismatch* est produite.
- DEF FN ne peut s'utiliser qu'en mode programme.
- L'utilisation d'une fonction par FN <nom de fonction> alors que cette fonction n'a pas été définie au préalable par DEF FN est interdit. Le message *Undefined user function* apparaît alors.
- Une fonction définie par le programmeur peut travailler sur des nombres ou sur des chaînes de caractères.

Exemples

```
DEF FN A$(P$,Q$,R$)=P$+Q$+R$
DEF FN E2(X,Y,Z)=X↑3+Y↑3+Z↑3
```

*** DEF USR [<numéro d'appel>] = <expression entière>**

Cette instruction sert à spécifier l'adresse de départ d'une routine écrite en langage machine. L'adresse de départ est le résultat de l'évaluation de <expression entière> et doit être comprise entre Ø et 65535 ; <numéro d'appel> est un nombre entier compris entre Ø et 9. Par défaut, <numéro d'appel> vaut Ø.

Un nombre quelconque de DEF USR peut être placé dans un programme pour redéfinir les adresses de départ des routines utilisées. Cela permet donc d'accéder à autant de routines en langage machine que nécessaire.

Exemple

```
DEF USR1=32ØØØ
```

*** VARPTR (<variable>) ou VARPTR (<n° de fichier>)**

VARPTR n'est pas une instruction mais une fonction. Elle est placée dans ce paragraphe pour la clarté de l'exposé. VARPTR (<variable>) renvoie l'adresse de début en mémoire de <variable> à condition que <variable> ait été définie avant l'appel de VARPTR.

Tous les types de variables sont acceptés.

Rappels

- Les variables entières sont stockées sur deux octets consécutifs, l'octet de poids faible en premier.
- Les variables réelles simple précision et double précision sont stockées sur respectivement 4 et 8 octets, octets de poids faibles en premier.
- Le descripteur d'une chaîne de caractères comporte 3 octets (longueur et adresse du premier caractère de la chaîne). C'est l'adresse de ce descripteur qui est renvoyée par VARPTR.
- Les éléments d'un tableau numérique occupent la place correspondant au type du tableau et sont rangés du bas de la mémoire vers le haut en partant du plus petit indice. Ainsi, pour un tableau de variables entières, A %(),VARPTR donne l'adresse de l'octet de poids faible de A %(O). Pour passer à l'indice suivant, il faut donc ajouter 2 à la valeur fournie par VARPTR.
- Dans le cas d'un tableau de chaînes de caractères, VARPTR renvoie l'adresse du descripteur de la première chaîne du tableau.

L'adresse fournie par VARPTR est un nombre entier compris entre −32768 et 32767. Dans le cas d'une adresse négative, l'adresse réelle est obtenue en ajoutant 65536.

VARPTR (<n° de fichier>) donne l'adresse du bloc descripteur du fichier spécifié.

*** PEEK(n)**

Cette fonction donne la valeur de l'octet stocké à l'adresse n en mémoire (n doit être compris entre −32768 et 65535).

Associée à VARPTR, la fonction PEEK permet de visualiser la façon dont les variables sont mémorisées.

Exemple

```
1Ø INPUT «DONNEZ UN NOMBRE ENTIER»;I %
2Ø ADR = VARPTR (I %)
3Ø PRINT «POIDS FAIBLE :»;PEEK(ADR)
3Ø PRINT «POIDS FORT :»;PEEK(ADR+1)
4Ø END
```

On peut ainsi vérifier que dans la case mémoire d'adresse ADR, il y a le reste de la division entière de A % par 256 (et donc l'octet de poids faible), alors que la case mémoire d'adresse ADR+1 contient le quotient de cette même division (donc l'octet de poids fort).

```
A % = 256 * PEEK (ADR + 1) + PEEK (ADR)
```

*** POKE <adresse>,<expression entière>**

L'instruction POKE permet d'accéder directement à la mémoire en écriture. Elle permet en effet d'affecter à la case mémoire d'adresse <adresse> la valeur de <expression entière>.

<adresse> doit être un entier compris entre −32768 et 65535, sachant que les adresses négatives correspondent à des adresses réelles égales à (adresse +65536) ; <expression entière> doit avoir une valeur entière comprise entre Ø et 255 (stockage sur un octet).

L'instruction POKE est généralement utilisée pour passer des paramètres à des sous-programmes écrits en langage machine. Il faut employer cette instruction avec précaution car elle permet de mettre n'importe quoi n'importe où et ce ne sera pas forcément apprécié par le système d'exploitation de la machine.

Exemple d'application : stockage d'un nombre entier en mémoire

```
1Ø DEF FN LOW (X %) = X %MOD256
2Ø DEF FN HIGH (X %) = X %\256
3Ø INPUT «DONNEZ UN NOMBRE ENTIER»;N %
4Ø INPUT «DONNEZ UNE ADRESSE»;ADR %
5Ø POKE ADR, FN LOW(N %)
6Ø POKE ADR+1, FN HIGH(N %)
7Ø END
```

2.1.3 Instructions d'usage général

*** BEEP**

Cette instruction génère un son de longueur fixe. Elle est équivalente à PRINT CHR$(7).

Remarque

BEEP réinitialise PLAY avec les valeurs par défaut (voir le chapitre concernant les instructions graphiques et sonores).

*** CLEAR [<espace chaîne>[, haut de la mémoire]]**

L'instruction CLEAR a pour effet de mettre toutes les variables numériques à Ø, toutes les variables alphanumériques à « » (chaîne vide) et de fermer tous les fichiers ouverts.

La syntaxe de CLEAR prévoit deux paramètres :

— <espace chaîne> fixe l'espace mémoire réservé aux chaînes de caractères. Par défaut cet espace mémoire est de 2ØØ octets, ce qui est souvent trop peu quand on travaille avec beaucoup de chaînes ; CLEAR permet donc de se réserver soi-même la place nécessaire ;

— <haut de la mémoire> est l'adresse maximale au-delà de laquelle l'interpréteur Basic n'a pas accès. En agissant sur le paramètre <haut de la mémoire> grâce à l'instruction CLEAR, le programmeur peut se réserver un espace mémoire protégé pour y mettre des sous-programmes en langage machine.

Exemples

CLEAR 5ØØ	efface toutes les variables et réserve 5ØØ octets pour les chaînes de caractères.
CLEAR	efface toutes les variables.
CLEAR, 16384	efface toutes les variables et interdit à l'interpréteur Basic les adresses supérieures à 16384.

*** END**

Cette instruction indique la fin de l'exécution du programme, ferme tous les fichiers ouverts et revient au mode direct (apparition du symbole OK et du curseur).

L'instruction END peut être placée n'importe où dans un programme pour en arrêter l'exécution. Sa présence est optionnelle à la fin d'un programme mais elle est vivement conseillée.

Exemple

```
1Ø  FOR I=1TO1Ø
2Ø  GOSUB 1ØØ
3Ø  NEXT I
4Ø  END
1ØØ PRINT «COUCOU»
11Ø RETURN
```

La présence de l'instruction END en ligne 4Ø est indispensable dans ce cas pour éviter qu'une erreur ne se produise *(Return without gosub)*.

*** REM <texte>**

Cette instruction permet de mettre des commentaires dans un programme. Tous les caractères situés à droite de REM sont ignorés par l'interpréteur Basic.

REM est non seulement utilisée pour commenter le programme mais aussi pour en clarifier et aérer la présentation. Il est en effet beaucoup plus facile de travailler sur un listing bien commenté et clair que de mettre au point un programme qui serait une simple juxtaposition d'instructions.

Des commentaires peuvent être placés à la fin d'une ligne de programme, précédés par une simple apostrophe.

Exemples

```
1Ø A=2Ø:REM commentaire
2Ø B=1Ø' commentaire
3Ø REM fin de programme
```

*** STOP**

Cette instruction provoque l'interruption de l'exécution d'un programme. Il y a retour au mode direct et affichage d'un message indiquant le numéro de la ligne où l'exécution s'est arrêtée *(Beak in nm)*.

Après une interruption par STOP, l'exécution peut être relancée par la commande CONT sous réserve qu'aucune modification n'ait été apportée au programme pendant l'interruption. L'instruction STOP peut être placée n'importe où dans le programme et sert ainsi de point d'arrêt pour la mise au point du programme.

Remarques

- Contrairement à l'instruction END, l'instruction STOP ne ferme pas les fichiers ouverts.
- L'instruction STOP est équivalente à la pression simultanée des touches CTRL et STOP.

*** WAIT <numéro de port>,<octet 1>[,<octet 2>]**

L'instruction WAIT interrompt l'exécution du programme tant que ne se présente pas sur le port d'entrée de numéro <numéro de port> un octet tel que le AND de cet octet et de <octet 1> donne une valeur non nulle.

Si <octet 2> est spécifié, l'octet lu sur le port fait d'abord l'objet d'un XOR avec <octet 2> avant d'être comparé à <octet 1>. Par défaut <octet 2> est pris nul.

<numéro de port>, <octet 1> et <octet 2> doivent être des nombres entiers compris entre Ø et 255.

2.1.4 Instructions d'entrée-sortie

Un micro-ordinateur, comme tout ordinateur, est une machine destinée au traitement de l'information. Le travail d'un micro-ordinateur comporte trois phases essentielles : l'acquisition des données, le traitement de ces données et l'affichage des résultats obtenus. Les phases d'acquisition des données et de sortie des résultats sont les phases de « dialogue » entre l'utilisateur d'un programme ou des périphériques entourant l'ordinateur et le programme lui-même. Ces phases nécessitent des instructions spéciales, appelées instructions d'entrée-sortie. Quelques fonctions interviennent aussi pour les entrées et les sorties. Elles seront donc également étudiées dans ce paragraphe.

INSTRUCTIONS D'ENTREE

* DATA <liste de constantes>

L'instruction DATA permet de stocker des constantes numériques ou alphanumériques. Ces constantes pourront ensuite être relues avec l'instruction READ puis affectées à des variables.

Une instruction DATA est une commande non exécutable et peut être placée n'importe où dans un programme. La liste de constantes peut être aussi longue que voulu dans la limite des 255 caractères accordés au maximum pour une ligne de programme.

Les constantes peuvent être numériques ou alphanumériques mais ne peuvent pas être des expressions numériques. Les guillemets ne sont pas nécessaires pour les constantes alphanumériques sauf dans le cas où une chaîne de caractères contient soit une virgule, soit deux points(:) soit des espaces significatifs.

La lecture des données par l'instruction READ se fait constante par constante en lisant les DATA par ordre croissant des numéros de ligne.

* READ <liste de variables>

L'instruction affecte à une liste de variables les constantes stockées avec des instructions DATA. Le type des variables doit correspondre avec le type des constantes lues.

Les lignes de DATA sont lues dans l'ordre croissant des numéros de ligne, en commençant au début du programme ou au numéro de ligne spécifié par une instruction RESTORE. Essayer de lire plus de données que ce qui est stocké provoque l'apparition du message *« Out of DATA »*.

Exemples

```
1Ø FOR I= 1 TO 5
2Ø READ PRENOM$
3Ø NEXTI
4Ø DATA ALAIN,JEAN,PAUL,DIDIER,PIERRE,JACQUES
```

Remarque

JACQUES ne sera jamais lu.

```
1Ø READ NOM$,NOTE %
2Ø PRINT «NOM:»;NOM$,«NOTE:»;NOTE %
3Ø DATA FDS,2Ø
```

Les lignes de DATA peuvent être placées n'importe où mais il est conseillé de les rassembler à la fin des programmes.

*** RESTORE [<numéro de ligne>]**

L'instruction RESTORE sert à réinitialiser (ou initialiser) le pointeur de DATA. Il peut en effet arriver que l'on désire lire plusieurs fois le même jeu de données ou que l'on veuille lire une ligne alors qu'il y a des DATA avant.

<numéro de ligne> peut être une constante ou une variable mais doit avoir une valeur entière qui correspond à une ligne existant dans le programme. Le pointeur de DATA sera placé sur la première ligne contenant des DATA à partir de <numéro de ligne>.

Si <numéro de ligne> est omis, le pointeur est placé sur la première ligne du programme contenant des données.

*** INPUT [«<message>»;] <liste de variables>**

L'instruction INPUT permet de fournir des données, à partir du clavier, au micro-ordinateur en cours d'exécution d'un programme. Un point d'interrogation est affiché, signifiant que la machine attend des données. Si un <message> est spécifié, ce message est affiché avant le point d'interrogation.

La liste de variables qui suit une instruction INPUT peut contenir un nombre quelconque de variables de types différents, séparées par une virgule. Les données entrées doivent être séparées par une virgule. L'entrée est validée en pressant la touche RC.

Si aucune donnée n'est fournie, les variables numériques seront mises à Ø et les chaînes de caractères seront égales à « » (chaîne vide). Si l'opérateur a fourni plus de donnés qu'il n'y a de variables dans la liste suivant l'instruction INPUT, les données supplémentaires sont ignorées. Le message *? Extra ignored* est affiché ; l'exécution du programme se poursuit. Si l'opérateur ne fournit pas assez de données, l'ordinateur affiche deux points d'interrogation, signifiant ainsi qu'il attend les données manquantes.

Les types des données introduites doivent correspondre aux

types des variables auxquelles ces données sont destinées. En cas d'erreur, le message *?Redo from start* est affiché. Toute l'entrée est alors à recommencer.

Les guillemets ne sont pas obligatoires pour les chaînes de caractères sauf si celles-ci comportent des virgules ou des espaces significatifs. L'utilisateur peut interrompre l'exécution du programme au niveau d'une instruction INPUT en pressant soit CTRL/C soit CTRL/STOP. Dans ce cas, il y a retour au mode direct et le symbole Ok réapparaît. La commande CONT permet de relancer l'exécution du programme à partir de l'instruction INPUT.

Exemples

```
INPUT «DONNEZ UN NOMBRE»;A
INPUT A %,B$,C!,D#
```

*** LINE INPUT [«<message>»;] <variable chaîne>**

L'instruction LINE INPUT permet d'affecter à une variable de type chaîne une chaîne de caractères quelconques (254 caractères au maximum). Contrairement à INPUT, l'instruction LINE INPUT n'affiche pas de point d'interrogation. Seul le message, s'il est spécifié, est affiché.

L'entrée est validée en pressant la touche RC.

La sortie d'un programme au niveau d'une instruction LINE INPUT se fait comme dans le cas de INPUT (CTRL/ou CTRL/STOP).

Exemple

```
1Ø LINE INPUT A$
2Ø PRINT A$
```

Si, à l'exécution, on tape 2*3*SIN(PI/2)+4, on obtient à l'affichage

```
2*3*SIN(PI/2)+4
OK
```

*** INPUT$ (X)**

INPUT$ est une fonction qui s'utilise généralement de la façon suivante : A$ = INPUT$(X). La machine attend alors que X caractères aient été tapés «au vol» sur le clavier. Ces X caractères forment

alors une chaîne de caractères qui est affectée à la variable A$. Le programme reprend son exécution.

Exemple

```
10 PRINT «PRESSEZ UNE TOUCHE»
20 A$ = INPUT$(1)
30 END
```

Remarques

- L'instruction INPUT$ permet la saisie de n'importe quel caractère du clavier sauf CTRL/STOP qui provoque l'arrêt du programme.
- Les caractères entrés ne sont pas affichés, INPUT$ sera donc très utile pour les problèmes de mots de passe.
- X doit être un nombre entier compris entre 1 et 255.

Exemple

```
10 PRINT «DONNEZ VOTRE MOT DE PASSE»
20 A$ = INPUT$(4)
30 IF A$ <> «PASS» THEN GOTO 10
40 END
```

*** INKEY$**

La fonction INKEY$ scrute brièvement le clavier sans attendre, à la différence de la fonction INPUT$, que des caractères soient tapés. INKEY$ s'emploie de la façon suivante : A$ = INKEY$. Si une touche est pressée au moment où la fonction INKEY$ est exécutée, le code du caractère correspondant est affecté à la variable chaîne A$, sinon A$ sera égal à « » (chaîne vide).

Remarques

- INKEY$ ne peut saisir qu'un caractère à la fois dans A$.
- Comme pour la fonction INPUT$, il n'y a pas d'écho du caractère frappé sur le téléviseur.
- La fonction INKEY$ est souvent employée dans les programmes de jeu. Placé dans la boucle principale du jeu, elle permet de lire « en permanence » le clavier pendant que le jeu se déroule.

Exemple

```
1Ø FOR I = 1 TO 1ØØØØ
2Ø A$ = INKEY$
3Ø IFA$ <> «» THEN GOTO 5Ø
4Ø NEXT I
5Ø PRINT «TEMPS DE REACTION:»;I
6Ø END
```

*** INP (<numéro de port>)**

La fonction INP renvoie l'octet lu sur le port d'entrée de numéro <numéro de port>, lequel doit être un nombre entier compris entre Ø et 255.

INSTRUCTIONS DE SORTIE

*** PRINT [<liste d'expressions>]**

L'instruction PRINT permet d'afficher les données sur l'écran à la position courante du curseur.

<liste d'expressions> est une liste d'expressions numériques ou alphanumériques (les chaînes de caractères doivent être placées entre guillemets). Les données affichées sont les valeurs prises par les différentes expressions composant <liste d'expressions>.

PRINT employé sans argument permet de sauter une ligne.

La position d'affichage de chacune des données est déterminée par la ponctuation qui la sépare de la donnée précédente dans <liste d'expressions> :

— Un point-virgule indique que la donnée doit être placée immédiatement après la dernière valeur affichée.

— Une virgule indique que la donnée doit être placée au début de la position de tabulation suivante (une position toutes les quatorze colonnes).

— Un ou plusieurs espaces ont le même effet qu'un point-virgule.

Si <liste d'expressions> est terminée par un point-virgule, la position courante du curseur est le premier espace suivant la dernière donnée affichée. Une virgule déplace la position courante

du curseur sur la position de tabulation suivante. Si aucune de ces deux ponctuations ne termine <liste d'expressions>, un caractère RC est envoyé à la console, ce qui a pour effet de positionner le curseur au début de la ligne suivante.

Si l'affichage de <liste d'expressions> nécessite plus d'une ligne, l'impression continue automatiquement à la ligne suivante.

Les valeurs numériques sont toujours suivies par un espace. Les nombres positifs sont précédés d'un espace et les nombres négatifs sont précédés par un signe —.

Remarque

En mode programmé comme en mode direct, un simple point d'interrogation peut remplacer le mot-clé PRINT.

Exemple

```
1Ø A=-5.3 : B=14.2 : C=-2E-4
2Ø PRINT A,
3Ø PRINT B
4Ø PRINT C;B;A;
5Ø PRINT «FIN»,
6Ø END
```

A l'exécution, on obtient :

```
-5.3^^^^^^^^^^^14.2
-2E-04^^14.2^-5.3 FIN
```

^ *représente un espace*

*** PRINT USING <chaîne de caractères> ; <liste d'expressions>**

Cette instruction est utilisée pour afficher des données numériques ou alphanumériques suivant un format particulier.

<liste d'expressions> comprend des expressions numériques ou alphanumériques, séparées les unes des autres par des points-virgules.

<chaîne de caractères> comporte les caractères spéciaux de formattage qui déterminent le champ et le format des données à afficher.

- affichage de chaînes de caractères

○ « ! » indique que seul le premier caractère de la chaîne spécifiée doit être affiché.

Exemple

```
A$ = «EDIMICRO»
OK
PRINT USING «!»;A$
E
OK
```

○ «&n espaces &» indique que 2+n caractères doivent être affichés. Si la chaîne est plus longue que le champ spécifié, les derniers caractères sont ignorés. Si le champ est plus long que la chaîne, elle est complétée par des caractères «espace».

Exemple

```
A$ = «ESSAI»
OK
PRINT USING «&^^&»;A$
ESSA
OK
PRINT USING «&&»;A$
ES
OK
```

^ *représente un espace*

○ «@» indique que toute la chaîne indiquée est affichée à la place du symbole.

Exemple

```
A$ = «ESSAI»
OK
PRINT USING «CET @ EST REUSSI»;A$
CET ESSAI EST REUSSI
OK
```

- 2e cas : affichage de valeurs numériques

○ « # » est utilisé pour représenter chaque chiffre du nombre à afficher. Le point décimal (.) peut être inséré à n'importe quelle position. Les nombres sont arrondis.

Exemple

```
PRINT USING «###.##»;10.2,2,3.456,.24
- 10.20^^2.00^^3.46^^0.24
OK
```

^ *symbolise un espace*

○ « + » est à placer au début ou à la fin de la chaîne de format pour positionner le signe réel du nombre à afficher.

Exemple

```
PRINT USING «+##.##»;1.25,-1.25
^+1.25^-1.25
OK
PRINT USING «##.##+»;1.25,-1.25
^1.25+^1.25-
OK
```

^ *symbolise un espace*

○ « − » placé à la fin de la chaîne de format indique que les nombres négatifs sont affichés avec le signe − en bout.

Exemple

```
PRINT USING «##.##-»;1.25,-1.25
^1.25^1.25-
OK
```

○ « ** » indique que les cases inoccupées sont remplacées par des astérisques.

Exemple

```
PRINT USING «**#.##»;1.25,-1.25
**1.25*-1.25
OK
```

○ « \\ » indique que le caractère « \ » complète le champ immédiatement à la gauche du nombre.

Exemple

```
PRINT USING «\\###.##»;23.56,-23.56
\\ 23.56\\-23.56
OK
PRINT USING «\\#-##.##-»;23.56,-23.56
\\ 23.56\\ 23.56-
OK
```

○ « ** \ » combinaison des deux effets.

Exemple

```
PRINT USING «**\##.##»;23.56
*\23.56
OK
```

○ « , » est utilisé pour la séparation de la valeur entière par des virgules tous les 3 chiffres.

Exemple

```
PRINT USING «####,.##»; 2356.7
2,356.70
OK
```

○ « ^^^^ » est utilisé pour la forme exponentielle des nombres.

Exemples

```
PRINT USING «##.##^^^^»;235.67
2.36E+02
PRINT USING «+##.##^^^^»;23.56,-23.56
+23.56E+00-23.56E+00
```

○ « % » si le nombre est plus grand que le champ qui lui est réservé, le signe % est affiché avant.

Exemple

```
PRINT USING «##.##»,235.67
%235.67
OK
PRINT USING «.##»;0.999
%1.00
OK
```

*** LPRINT et LPRINT USING**

Ce sont les mêmes instructions que PRINT et PRINT USING, mais l'impression se fait sur une imprimante.

*** TAB(N)**

La fonction TAB permet de positionner horizontalement le curseur sur n'importe laquelle des 4Ø positions standard d'écriture.

N est un nombre entier positif compris entre Ø et 255. N représente le numéro de la colonne d'affichage (module 4Ø).

La colonne Ø est située à l'extrême-gauche de l'écran, la colonne 39 est située à l'extrême-droite.

La fonction TAB ne peut être utilisée qu'avec les instructions PRINT et LPRINT.

*** OUT <numéro de port>,<expression entière>**

L'instruction OUT permet d'envoyer un octet sur un port de sortie (pour commander un périphérique quelconque : écran, imprimante,..).

<numéro de port> et la valeur de <expression entière> doivent être des nombres entiers compris entre Ø et 255.

2.1.5 Boucles et branchements

BOUCLES

Certains calculs ou traitements doivent être exécutés plusieurs fois. Il est alors intéressant de créer une boucle qui sera parcourue

autant de fois que nécessaire. Les instructions FOR, TO, STEP, NEXT permettent la création de telles boucles.

La syntaxe est la suivante :

FOR <variable numérique> = X TO Y [STEP Z]
traitement
NEXT [<variable numérique>]

La variable numérique qui suit le mot-clé FOR est appelée variable de contrôle de la boucle. Cette variable joue le rôle de compteur. Elle peut être de type entier, réel simple ou double précision, mais seul le choix de valeurs entières pour <variable numérique> et le pas Z optimise la vitesse d'exécution de la boucle. X est la valeur initiale du compteur, Y sa valeur finale.

L'expression STEP Z est facultative. Par défaut, le pas (STEP) Z est pris égal à 1.

<variable numérique> peut être omis après NEXT. Dans ce cas, chaque NEXT est associé au premier FOR qui le précède.

Les boucles FOR-NEXT peuvent être imbriquées, le nombre d'imbrications ne dépendant que de la taille de mémoire disponible sur le micro-ordinateur. Si toutes les boucles imbriquées ont un point de sortie commun, un seul NEXT est nécessaire.

Comment fonctionne une boucle FOR-NEXT ?

— la variable de contrôle prend la valeur X,
— la boucle est exécutée,
— lorsque le programme arrive sur l'instruction NEXT, la variable de contrôle est incrémentée (variable de contrôle = variable de contrôle + Z),
— si la variable de contrôle est plus petite que la valeur finale Y (cas d'un pas Z positif) ou si elle est plus grande (dans le cas d'un pas Z négatif), il y a retour au début de la boucle ; sinon le programme « sort » de la boucle et passe aux instructions suivantes.

Cela implique donc que le corps de la boucle soit toujours parcouru au moins une fois.

Remarques

- Plusieurs boucles peuvent être imbriquées mais non entrelacées.

• On peut sortir d'une boucle par une instruction GOTO mais il est interdit de rentrer au milieu d'une boucle (erreur NEXT without FOR).

• Il est déconseillé de se servir dans la suite du programme de la variable ayant servi de compteur pour une boucle. Cela peut entraîner des erreurs assez fines à retrouver ensuite.

Exemples

```
1Ø FOR I=1 TO 1Ø
2Ø PRINT I
3Ø NEXT I
```

```
1Ø FOR I=1Ø TO 1 STEP -1
2Ø FOR J=1 TO 1Ø
3Ø PRINT I*J
4Ø NEXT
```

INSTRUCTIONS DE CONTRÔLE ET DE BRANCHEMENT

Bien que les instructions de contrôle soient différentes de celles de branchement, il est plus aisé de les étudier conjointement. En effet, dans la majeure partie des cas, une instruction de contrôle se trouve associée à une instruction de branchement.

Une instruction de contrôle réalise, comme son nom l'indique, un test ou un contrôle.

Une instruction de branchement permet de faire une rupture de séquence dans le déroulement d'un programme. Normalement un programme est effectué de façon séquentielle, c'est-à-dire par ordre de numéros de lignes croissants. Cette façon de procéder est modifiable par une instruction de branchement, qui indique à l'ordinateur que la prochaine ligne à exécuter n'est pas la ligne suivante, mais la ligne qui a le numéro indiqué par l'instruction de branchement.

Ces deux types d'instructions sont fondamentaux pour la programmation car ils permettent d'écrire des programmes capables de prendre des décisions.

* GOTO <numéro de ligne>

Cette instruction effectue un branchement inconditionnel à la

ligne de programme <numéro de ligne>, lequel <numéro de ligne> doit correspondre au numéro d'une ligne existant réellement, sinon le message *Undefined line number* apparaît.

Le Basic, par sa nature même, oblige les programmeurs à employer l'instruction GOTO. Attention, tout de même ; un usage abusif de cette instruction est vivement déconseillé car il mène très vite à des programmes inextricables dont la mise au point est alors très difficile. Contrairement à ce que pensent beaucoup de gens, on peut écrire des programmes propres et structurés en Basic ; il suffit de s'en donner la peine.

GOTO est aussi très utile en mode direct pour essayer des parties d'un programme en cours de mise au point.

Exemple

```
1Ø PRINT « Connaissez-vous le mouvement perpétuel ? »
2Ø GOTO 1Ø
```

*** GOSUB <numéro de ligne>**
RETURN

L'analyse d'un problème permet souvent de le diviser en plusieurs petits problèmes pouvant être résolus séparément, la mise en commun des solutions de chacun d'eux permettant ensuite de résoudre le problème initial. On peut donc écrire un programme principal et plusieurs « petits » programmes appelés « sous-programmes » (en fait souvent plus longs que le programme principal). Les sous-programmes sont appelés les uns après les autres par le programme principal afin de résoudre les petits problèmes, le programme principal se chargeant de trouver la solution globale.

Cette façon de procéder est appelée programmation structurée. Elle permet, quel que soit le langage de programmation utilisé, d'écrire des programmes modulaires, simples, clairs, facilement mis au point module par module et aisément modifiables.

L'instruction d'appel d'un sous-programme en Basic est l'instruction GOSUB n où n est le numéro de la première ligne du sous-programme. Le sous-programme doit se terminer par l'instruction RETURN afin de redonner le contrôle au programme principal dès qu'il a fini son travail.

Remarques

• Les sous-programmes peuvent appeler eux-mêmes d'autres sous-programmes, le nombre d'appels n'étant limité que par la place mémoire maximale accordée à la pile de l'ordinateur (la pile est la zone mémoire où est stockée le numéro de la ligne à partir de laquelle l'exécution du programme principal reprend après un RETURN).

• On peut sortir d'un sous-programme par une instruction de branchement inconditionnel mais on ne peut pas entrer directement dans le corps d'un sous-programme (erreur RETURN without GOSUB).

Exemple

```
10 PI=3.14
20 INPUT «RAYON»;R
30 GOSUB 100:REM calcul de la circonférence
40 GOSUB 200:REM calcul de la surface
50 PRINT «circonférence= »;C
60 PRINT «SURFACE= »;S
70 END
100 REM CIRCONFERENCE
110 C=2*PI*R
120 RETURN
200 REM SURFACE
210 S=PI*R↑2
220 RETURN
```

GOTO

*** ON <expression numérique> GOSUB <numéro de ligne>[,<numéro de ligne>]...]**

Ces deux instructions permettent un branchement calculé ; l'une vers un numéro de ligne (GOTO), l'autre vers le début d'un sous-programme.

Si <expression numérique>=1, le GOTO ou le GOSUB s'effectuera avec le premier numéro de la liste ; si <expression numérique>=2, le branchement se fera avec le deuxième numéro, etc.

Si <expression numérique>=0 ou si <expression numérique> est supérieur au nombre de numéros de lignes spécifiés, le programme continue à s'exécuter à partir de la ligne suivante.

Une erreur apparaît (*Illegal function call*) si la valeur de <expression numérique> est négative ou supérieure à 255.

Exemple

Le programme suivant détermine si une année est bissextile.

```
10 INPUT «ANNEE»;AN %
20 PRINT «L'ANNEE»;AN %
30 ON (AN %MOD4+1) GOSUB 60,80,80,80
40 PRINT «BISSEXTILE»
50 END
60 PRINT «EST»;
70 RETURN
80 PRINT «N'EST PAS»;
90 RETURN
```

*** IF <expression> THEN <instruction ou numéro de ligne>**
[ELSE <instruction ou numéro de ligne>]
IF <expression> GOTO <numéro de ligne>
[ELSE <instruction ou numéro de ligne>

La structure IF/THEN/ELSE permet d'effectuer des choix en fonction du résultat d'un test ou d'un calcul. Le programme ci-dessous montre la plupart des cas d'application de cette structure.

```
10 INPUT «A=»;A
20 INPUT «B=»;B
30 PRINT
40 IF A=B THEN PRINT «A EGAL B» ELSE PRINT «A EST
DIFFERENT DE B»
50 PRINT
60 IF A THEN PRINT «A EST NUL» :PRINT
70 IF A*B<0 THEN IF A<B THEN PRINT «A EST NEGATIF, B EST
POSITIF» ELSE PRINT «A EST POSITIF, B EST NEGATIF»
80 PRINT
90 IF A>B OR A<0 GOTO 100 ELSE GOTO 200
100 IF A<B AND B>0 THEN 10
200 GOTO 10
```

— Les lignes 1Ø et 2Ø permettent d'entrer les deux nombres A et B sur lesquels vont porter les tests.

— Les lignes 3Ø, 5Ø et 8Ø ont pour but d'aérer la présentation des résultats sur l'écran.

— La ligne 4Ø est une application typique de l'instruction de test et n'a pas besoin d'explication supplémentaire.

— La ligne 6Ø teste si A est nul (IF A est équivalent à IF(A=Ø). Si c'est le cas, le message A EST NUL apparaît, sinon il y a passage à la ligne 7Ø.

— La ligne 7Ø présente une mise en casacade de plusieurs tests :

- si le produit A*B est positif, il y a passage à la ligne 8Ø sans exécuter le reste de la ligne 7Ø ;
- si le produit A*B est négatif alors on teste si A est plus petit que B. Si c'est le cas, on en déduit que A est négatif et B positif, sinon c'est A qui est positif et B négatif.

— En ligne 9Ø le test porte sur une expression faisant intervenir des opérateurs logiques et relationnels. Si A est plus grand que B *ou* si A est négatif, l'expression logique est dite « vraie » (valeur −1) et le branchement se fait à la ligne 1ØØ. Si A est inférieur ou égal à B *et* A est positif ou nul (expression logique contraposée de l'expression logique précédente) alors il y a branchement à la ligne 2ØØ.

— Un raisonnement identique s'applique à la ligne 1ØØ.

2.1.6 Les interruptions du MSX-Basic

Le MSX-Basic possède un jeu d'instructions spécialisées qui permettent un branchement automatique vers des sous-programmes d'interruption chaque fois que la pression d'une touche ou la validation d'une fonction spéciale génère une interruption.

Ce déroutement qui est prioritaire sur toutes les autres instructions Basic peut être autorisé, interdit ou inhibé temporairement grâce à des instructions spéciales. Ainsi le programmeur peut affecter à chaque interruption le niveau de priorité qu'il désire. Il lui suffit en effet d'insérer dès le début du sous-programme d'une interruption très prioritaire, les commandes de mémorisation des interruptions moins prioritaires. A la fin du traitement de l'interruption prioritaire (c'est-à-dire juste avant le RETURN), il lui

suffit de revalider les autres interruptions. Une interruption, qui aurait donc été mémorisée pendant le précédent traitement, peut alors être prise en compte.

Le MSX-Basic possède cinq instructions d'interruption :

ON INTERVAL < paramètres> GOSUB
ON STOP GOSUB
ON KEY GOSUB
ON STRIG GOSUB
ON SPRITE GOSUB

Les cinq instructions de validation, masquage, mémorisation associées sont :

INTERVAL	ON/OFF/STOP
STOP	ON/OFF/STOP
KEY(N)	ON/OFF/STOP
STRIG(N)	ON/OFF/STOP
SPRITE	ON/OFF/STOP

C'est la première fois qu'un Basic dispose d'instructions aussi puissantes pour traiter des interruptions. Cela ouvre la voie à des applications très sophistiquées : temps réel, jeux vidéo complexes...

Nous n'étudierons dans ce paragraphe que les interruptions gérées par les commandes INTERVAL et STOP. Les autres interruptions seront étudiées dans des paragraphes particuliers (gestion du clavier, gestion des manettes de jeux, gestion des sprites).

*** ON INTERVAL = <intervalle de temps> GOSUB <numéro de ligne>**

Cette instruction signifie que tous les <intervalle de temps> un branchement automatique sera fait au sous-programme commençant à la ligne <numéro de ligne>. Le branchement n'est réellement effectué que si l'interruption est autorisée par INTERVAL ON.

Les branchements ne fonctionnent qu'en mode programme. Une interruption est générée approximativement toutes les <intervalle de temps>/50 secondes.

Quand un déroutement est validé, un INTERVAL STOP est effectué (on évite ainsi les problèmes de réentrance et de bouclage infini).

En fin de sous-programme un INTERVAL ON est fait automatiquement sauf si un INTERVAL OFF a été fait de façon explicite dans le sous-programme d'interruption.

* INTERVAL ON/OFF/STOP

INTERVAL ON autorise les déroutements consécutifs à l'apparition d'une interruption timer.

INTERVAL OFF inhibe les interruptions. Il n'y a pas mémorisation.

INTERVAL STOP masque les interruptions mais il y a mémorisation. Si une interruption a été générée, dès que l'instruction INTERVAL ON est rencontrée, le sous-programme d'interruption spécifié par ON INTERVAL GOSUB est exécuté.

Exemple

```
 1Ø REM CHRONOMETRE
 2Ø S=Ø
 3Ø ON INTERVAL=5Ø GOSUB 1ØØ
 4Ø INTERVAL ON
 5Ø FOR I %=1 TO 1ØØØØØ
 6Ø PRINT I %;
 7Ø NEXT I %
 8Ø END
1ØØ CLS:S=S+1
11Ø BEEP
12Ø PRINT S;
13Ø IF S>1 THEN PRINT «SECONDES» ELSE PRINT «SECONDE»
14Ø RETURN
```

* ON STOP GOSUB <numéro de ligne>

Normalement la pression simultanée des touches CTRL et STOP provoque l'arrêt de l'exécution du programme et le retour au mode direct. L'instruction ON STOP GOSUB permet de dérouter le programme vers un sous-programme d'interruption quand CTRL et STOP sont pressées.

Les branchements ne fonctionnent qu'en mode programme. Quand un déroutement est validé ; un STOP STOP est effectué automatiquement. En fin de sous-programme, un STOP ON est fait automatiquement sauf si un STOP OFF figure de façon explicite dans le sous-programme d'interruption.

* STOP ON/OFF/STOP

STOP ON autorise les déroutements consécutifs à l'apparition d'une interruption due à la pression simultanée de CTRL et STOP.

STOP OFF interdit les interruptions. Il n'y a pas mémorisation.

STOP STOP inhibe les interruptions, met en mémoire la demande. Les interruptions seront validées dès qu'un STOP ON sera rencontré.

Le petit programme de jeu ci-dessous met en application l'instruction ON STOP GOSUB. Le jeu consiste à presser simultanément sur CTRL et sur STOP lorsque le compteur qui tourne sur l'écran passe par une valeur multiple de 5. Le compteur ne va que jusqu'à 5Ø, il faut donc réagir vite. Bien sûr il est interdit de maintenir en permanence les touches CTRL et STOP enfoncées sinon le jeu perd tout son intérêt.

```
 1Ø REM REFLEXE
 2Ø STOP ON
 3Ø ON STOP GOSUB 2ØØ
 4Ø CLS:A=1
 5Ø FOR I=1 TO 5Ø:REM COMPTEUR
 6Ø IF A =ØTHEN GOTO 15Ø:REM GAGNE
 7Ø PRINT I;
 8Ø FOR T=1 TO 5Ø:NEXT T:REM TEMPO
 9Ø PRINT CHR$(13);
1ØØ NEXT I
11Ø PRINT «PERDU»
12Ø END
15Ø PRINT:PRINT «GAGNE»
16Ø END
2ØØ IF I MOD5=Ø THEN A=Ø
21Ø RETURN
```

2.1.7 Traitement des erreurs

Lors de l'exécution d'un programme, si un calcul ou une instruction ne peut pas être exécuté, une *erreur* est produite. Le déroulement du programme est alors interrompu et le message d'erreur prévu par le système d'exploitation est affiché sur l'écran : division par Ø, dépassement de capacité, appel illégal d'une

fonction, erreur de syntaxe... Vous trouverez en annexe la liste complète des messages et des codes d'erreur du MSX-Basic.

Le programme peut désirer traiter lui-même le cas d'erreur pour personnaliser les messages d'erreur, par exemple, ou bien pour protéger son programme tout en évitant que son exécution ne soit interrompue par une erreur. Pour cela il dispose de trois instructions : ON ERROR GOTO, RESUME et ERROR.

*** ON ERROR GOTO <numéro de ligne>**

Si cette instruction est rencontrée dans le programme (il est souhaitable que ce soit à l'une des premières lignes du programme), il y aura branchement automatique vers une routine de traitement d'erreur commençant à la ligne <numéro de ligne>

L'interruption générée par l'apparition d'une erreur est prioritaire sur toutes les autres interruptions (INTERVAL, STOP, KEY, STRIG, SPRITE) et elle est non masquable. Le système met aussitôt à jour deux variables, ERR et ERL, qui pourront être utilisées par le programmeur pour savoir d'où vient l'erreur et quelle est sa nature.

ERR contient le code de l'erreur (voir annexe).

ERL contient le numéro de la ligne où l'erreur s'est produite.

Remarques

- L'instruction ON ERROR GOTO ∅ provoque un traitement normal des erreurs par l'interpréteur Basic : arrêt du programme, message d'erreur du système.
- Si une erreur se produit dans la routine de traitement d'erreur, cette erreur est prise en compte par le système.

*** RESUME [NEXT]**
RESUME [<numéro de ligne>]

Si l'on désire que l'exécution du programme puisse reprendre après le traitement d'une erreur, il est nécessaire d'employer l'instruction RESUME à la fin du programme d'erreur.

— RESUME ou RESUME ∅ fait reprendre l'exécution du programme à partir de la ligne où s'était produite l'erreur. L'instruction ou le calcul ayant provoqué l'erreur est effectué à nouveau.

— RESUME <numéro de ligne> provoque la reprise du déroulement à partir de la ligne spécifiée.

— RESUME NEXT fait poursuivre l'exécution du programme à partir de la première ligne suivant la ligne où s'était produite l'erreur.

L'instruction RESUME ne peut être employée que dans le cadre d'une routine de traitement d'erreur.

Exemple

Le programme que nous avions donné précédemment pour vérifier si une année est bissextile peut être réécrit de la façon suivante :

```
 5 ON ERROR GOTO 6Ø
1Ø INPUT «ANNEE»;AN%
2Ø PRINT «L'ANNEE»;AN%
3Ø ON (AN%MOD4)/(AN%MOD4)GOSUB 8Ø
4Ø PRINT «BISSEXTILE»
5Ø END
6Ø PRINT «EST»
7Ø RESUME NEXT
8Ø PRINT «N'EST PAS»;
9Ø RETURN
```

*** ERROR <expression entière>**

Cette instruction est utilisée pour simuler l'apparition d'erreurs et pour permettre la définition de codes d'erreur personnalisés.

<expression entière> doit avoir une valeur comprise entre Ø et 255. Si cette valeur correspond à un code déjà attribué par le système MSX, le message correspondant du système sera affiché. Pour définir ses propres codes d'erreur, il faut donc prendre des codes plus grands que tous ceux utilisés par le MSX-Basic.

Si une instruction ERROR simule une erreur pour laquelle aucun message d'erreur n'a été défini, le message *Unprintable essor* apparaît.

L'exécution d'une instruction ERROR pour laquelle il n'y a pas de routine de traitement provoque l'arrêt du programme et l'apparition du message *Unprintable error* si le code ne correspond pas à

une erreur du MSX-Basic (sinon c'est le message correspondant à l'erreur produite qui est affiché.

Exemple

```
 5 ON ERROR GOTO 1ØØ
1Ø INPUT «DONNEZ UN NOMBRE QUELCONQUE»;N
2Ø IF N=Ø THEN ERROR 1ØØ
3Ø PRINT «N EST DIFFERENT DE Ø»
4Ø END
1ØØ IF ERR=1ØØ THEN PRINT «N EST NUL».
11Ø RESUME 4Ø
```

2.1.8 Manipulation de fichiers

Un ordinateur est amené à dialoguer avec un certain nombre de périphériques : écran, imprimante, magnétophone à cassettes, lecteurs de disquettes, etc. Des fichiers de données ou des fichiers de programmes peuvent ainsi être transférés de l'ordinateur vers un périphérique et réciproquement (*exemple :* sauvegarde d'un programme sur une cassette magnétique).

En MSX-Basic le transfert de données nécessite l'ouverture préalable de canaux de communication entre l'ordinateur et les périphériques concernés. L'instruction dont le rôle est d'ouvrir ces canaux est l'instruction OPEN. Une fois le transfert terminé, les canaux pourront être refermés par l'instruction CLOSE.

Nous n'étudierons dans ce paragraphe que les instructions permettant la gestion de fichiers séquentiels. Le mode séquentiel est en effet le seul mode de gestion possible pour des fichiers stockés sur bande magnétique.

Rappels concernant les fichiers séquentiels

— Dans un fichier séquentiel, les données sont stockées les unes derrière les autres au fur et à mesure de leur arrivée. L'écriture d'une nouvelle donnée ne peut se faire qu'à la fin du fichier.

— Pour lire une donnée, il est nécessaire de les lire toutes depuis le début du fichier jusqu'à ce que la donnée désirée soit trouvée (accès séquentiel).

— Il est impossible de modifier une donnée située au milieu du

fichier sans avoir recours à des artifices du type réécriture de tout le fichier.

— La fin du fichier est marquée par un caractère spécial (CTRL/Z, code 26).

— A chaque fichier ouvert est associée une zone de mémoire réservée (mémoire tampon ou *buffer*). Toutes les données en transit entre l'ordinateur et le périphérique passent par ce buffer. Le transfert réel n'a lieu que lorsque le buffer est plein. Le transfert se fait donc par blocs de donnés.

*** OPEN «type de périphérique [<nom de fichier>]» [FOR <mode>] AS [#] <numéro de fichier>**

L'instruction OPEN permet d'ouvrir un canal entre l'ordinateur et le périphérique spécifié. Cette instruction réserve un buffer tampon pour les données et précise le mode de transfert.

<type de périphérique> peut être :
— CAS : magnétophone à cassettes
— CRT : écran basse résolution (24 lignes de 40 caractères)
— GRP : écran graphique
— LPT : imprimante

Cette liste n'est pas exhaustive car de nombreuses extensions peuvent être ajoutées aux machines MSX.

<nom du fichier> est facultatif. Il est surtout utilisé pour différencier des fichiers différents lors de leur sauvegarde sur cassette.

<mode> définit le mode d'accès au canal et le sens du transfert.

Il peut être :

— OUTPUT accès séquentiel en sortie (ordinateur vers périphérique)

— INPUT accès séquentiel en entrée (périphérique vers ordinateur)

— APPEND accès séquentiel en entrée et sortie (les données entrées sont ajoutées à la fin du fichier)

<numéro de fichier> est une expression entière dont la valeur doit être comprise entre Ø et le nombre maximum de fichiers pouvant être ouverts simultanément (ce nombre est donné par la valeur de la variable MAXFILES, voir plus loin). <numéro de

fichier> est associé au fichier tant que celui-ci est ouvert ; il sera utilisé dant toutes les instructions d'accès au fichier.

*** CLOSE [[#] <numéro de fichier> [, <nom de fichier>]]**

Cette instruction ferme tous les fichiers spécifiés et libère les buffers qui leur étaient associés.

CLOSE sans argument ferme tous les fichiers

Remarques

- L'emploi de CLOSE est obligatoire dès qu'un fichier n'est plus employé, car c'est cette instruction qui provoque le transfert des dernières données stockées dans le buffer, même si celui-ci n'est pas plein. Un caractère CTRL/Z (code ASCII 26) est placé automatiquement à la fin du fichier.
- END ferme tous les fichiers ouverts.
- STOP ne ferme pas les fichiers ouverts.

***MAXFILES = <expression entière>**

La valeur de la variable MAXFILES spécifie le nombre maximum de fichiers pouvant être ouverts simultanément en mémoire.

<expression entière> doit prendre une valeur entière comprise entre Ø et 15. Par défaut MAXFILES = 1.

Si MAXFILES = Ø, seules les commandes SAVE et LOAD sont acceptées par le Basic.

Remarque

La sauvegarde et le chargemant de programme à l'aide d'instruction du type SAVE et LOAD ne nécessitent pas l'ouverture ni la fermeture de canaux spéciaux par OPEN et CLOSE.

Exemple

```
MAXFILES = 2
OPEN «CAS : ESSAI. BAS» FOR OUTPUT AS # Ø
CLOSE Ø
```

*** EOF (<numéro de fichier>)**

Cette fonction renvoie − 1 si la fin d'un fichier séquentiel est atteinte lors d'une lecture par INPUT#, LINE INPUT# ou INPUT$(#).

*** INPUT# <numéro de fichier>, <liste de variables>**

Cette instruction permet, comme INPUT, de lire des données ; mais la lecture se fait non pas sur le clavier mais dans le fichier dont le numéro est spécifié. Les données lues sont assignées aux variables figurant dans <liste de variables>. Les types des données lues doivent correspondre aux types des variables de la liste. L'instruction INPUT# ne fait pas afficher un point d'interrogation puisque la lecture a lieu dans un buffer.

Pour les variables numériques, le premier caractère rencontré autre qu'un espace, un retour chariot RC, ou un saut de ligne LF est considéré comme le début d'un nombre. Le nombre se termine dès qu'une virgule ou l'un des trois séparateurs cités précédemment est rencontré.

Pour les données alphanumériques, si le premier caractère d'une chaîne n'est pas un guillemet, la lecture de la chaîne se poursuit, à concurrence de 255 caractères, jusqu'à ce qu'une virgule, un RC ou un LF soit rencontré.

Si le premier caractère d'une chaîne est un guillemet, la lecture de la chaîne se poursuit jusqu'au guillemet suivant. Si en cours de lecture de donnée le caractère de contrôle CTRL/Z est lu, la saisie est terminée.

Exemple

```
10 OPEN «CAS : ELEVE» FOR INPUT AS#1
20 INPUT#1, NOM$, NOTE, CLASSEMENT
30 PRINT NOM$; « :», NOTE; «:»;CLASSEMENT
40 CLOSE 1
50 END
```

*** LINE INPUT # <numéro de fichier>, <variable chaîne>**

Cette instruction permet de lire une chaîne de caractères sans tenir compte des séparateurs autres que *retour chariot* (code 13). 254 caractères au maximum peuvent être ainsi lus.

LINE INPUT# est souvent utilisé pour lire d'un seul coup l'enregistrement d'un fichier, cet enregistrement étant composé de plusieurs champs séparés par une virgule (la virgule est considérée comme un séparateurs pour INPUT#).

***INPUT$ (n, [#] <numéro de fichier>)**

Cette fonction permet de lire une chaîne de n caractères lus dans le fichier spécifié. Tous les caractères sans exception (sauf CTRL/Z qui interrompt la saisie) sont acceptés.

n doit être un nombre entier compris entre Ø et 255.

*** PRINT # <numéro de fichier>, <expression ou liste de variables>**
PRINT # <numéro de fichier> USING <chaîne de caractères> ; <liste d'expressions>

Ces deux instructions permettent d'écrire des données dans un fichier. Les caractéristiques de leurs arguments sont identiques à celles des arguments de PRINT et PRINT USING

Remarque

Le fichier spécifié doit avoir été ouvert au préalable par OPEN en mode OUTPUT ou APPEND.

2.2 LES FONCTIONS DU MSX-BASIC

Les fonctions du MSX-Basic peuvent être réparties en trois grands groupes.

— Les fonctions d'usage général

BASE	INKEY$	SPC
BIN$	INP	TAB
CDBL	INPUT$	TIME
CINT	INT	USR
CSNG	LPOS	VARPTR
FIX	OCT$	
FRE	PEEK	
HEX$	RND	

— Les fonctions de calcul

ABS	LOG	TAN
ATN	SGN	
COS	SIN	
EXP	SQR	

— Les fonctions destinées au traitement de chaînes de caractères

ASC	MID$
CHR$	RIGHT$
INSTR	SPACE$
LEFT$	STR$
LEN	STRING$

Certaines de ces fonctions ont déjà été introduites, quand cela était nécessaire pour l'étude d'instructions particulières. Nous donnerons pour ces fonctions les références des paragraphes correspondants.

2.2.1 Les fonctions d'usage général

*** BASE (<nombre entier>)**

Cette fonction renvoie l'adresse de début en mémoire des zones de stockage utilisées pour les données servant à l'affichage ; <nombre entier> doit être un entier compris entre 0 et 18.

En mode texte 40 caractères :

N = Ø	buffer texte
N = 2	générateur de caractères

En mode texte 32 colonnes

N = 5	buffer texte
N = 6	table des couleurs
N = 7	générateur de caractères
N = 8	affectation des sprites
N = 9	dessin des sprites

En mode haute résolution (256 × 192)

N = 1Ø	buffer texte
N = 11	table des couleurs

N = 12	générateur de caractères
N = 13	affectation des sprites
N = 14	dessin des sprites

En mode base résolution (64 × 48)

N = 15	buffer texte
N = 16	générateur de caractères
N = 17	affectation des sprites
N = 18	dessin des sprites

Cette fonction n'est véritablement utile que pour les programmeurs en langage machine.

*** BIN$ (n)**

Cette fonction donne une chaîne correspondant à l'équivalent binaire du nombre décimal n, lequel doit être compris entre − 32768 et 65535. Si n est négatif, BIN$ effectue le complément à 2 de n avant de faire la conversion (*i.e.* BIN$ (65536− n)).

*** CDBL (X)**
CINT(X) voir paragraphe 1.5.1Ø
CSNG (X)

*** FIX (X)**

FIX renvoie la partie entière de X par troncature de la partie fractionnaire.

FIX (X) est strictement équivalent à SGN(X) * INT (ABS(X)). La différence avec INT est que FIX ne donne pas, si X est négatif, le premier nombre entier inférieur à X.

FIX (− 9.8) = − 9
INT (− 9.8) = 1Ø

La différence avec CINT se situe au niveau de la représentation du résultat. FIX(X) est un nombre en double précision alors que CINT(X) est un entier.

*** FN <nom de la fonction> [(<liste de paramètres>)]**

Cette fonction associée à <nom de la fonction> sert à appeler une fonction définie au préalable avec DEF FN (voir paragraphe 2.1.2)

*** FRE (Ø)**
FRE (« »)

FRE (Ø) donne le nombre d'octets de mémoire encore disponibles.

FRE (« ») donne le nombre d'octets encore libres dans la zone réservée pour les chaînes de caractères (fixée à 2ØØ octets à l'origine, cette zone peut être agrandie avec l'instruction CLEAR).

*** HEX$ (n)**

Cette fonction donne une chaîne correspondant à l'équivalent hexadécimal du nombre décimal n, lequel doit être compris entre – 32768 et 65535. Si n est négatif, c'est HEX$ (65536 – n) qui est calculé.

*** INKEY$ voir paragraphe 2.1.4**
INP
INPUT$

*** INT (X)**

Cette fonction donne la partie entière du nombre x.

*** LPOS (X)**

LPOS donne la position de la tête de l'imprimante sur la ligne courante ; x est un argument fictif qui peut prendre n'importe quelle valeur.

*** OCT$ (n)**

Cette fonction donne une chaîne correspondant à l'équivalent octal du nombre décimal n, lequel n doit être compris entre – 32768 et 65535. Si n est négatif, c'est OCT$ (65536 – n) qui est calculé.

*** PEEK (<adresse>)**

Voir paragraphe 2.1.2

*** RND (n)**

La fonction RND donne un nombre aléatoire compris entre Ø et 1. Si n>Ø, le nombre est pris dans une séquence pseudo-aléatoire (cette séquence est la même à chaque exécution du programme). Si

n = Ø, RND renvoie le dernier nombre tiré. Si n<Ø, RND donne un nombre qui est fonction de n. On obtiendra donc toujours le même nombre pour des valeurs égales de n.

Pour obtenir un générateur de nombres réellement aléatoires, il faut associer la fonction RND avec la fonction TIME (ci-après).

*** SPC (X)**

Cette fonction imprime X espaces sur l'écran ou sur l'imprimante ; X, automatiquement converti en entier, doit avoir une valeur comprise entre Ø et 255.

*** TAB (X)**

Voir paragraphe 2.1.4

*** TIME**

TIME renvoie la valeur d'une variable système qui est mise à jour tous les cinquantièmes de seconde.

Il est possible de remettre à zéro l'horloge interne en faisant TIME = Ø. Cela permet de chronométrer des événements : durée d'une boucle, temps de réaction d'un joueur...

La fonction TIME permet aussi d'obtenir des séquences de nombres véritablement aléatoires.

Exemples

```
1Ø A = TIME
2Ø PRINT A, RND (A)
3Ø GOTO 1Ø
1Ø A = — TIME
2Ø PRINT A, RND (A)
3Ø GOTO 1Ø
```

*** USR [<nombre entier>] (X)**

Cette fonction permet l'appel d'un sous-programme, écrit en langage machine, dont l'adresse a été définie par une instruction DEF USR (voir paragraphe 2.1.2).

<nombre entier> doit être un entier compris entre Ø et 9. Par défaut il est pris égal à Ø. Il correspond au numéro de la routine spécifié dans DEFUSR.

X est le paramètre passé à la routine en assembleur. Ce paramètre et accessible par le sous-programme en langage machine de la façon suivante.

— S'il s'agit d'un entier :

la mémoire F663H contient la valeur 2
le paramètre se trouve en F7F8H (poids faible)
F7F9H (poids fort)

— S'il s'agit d'un réel en simple précision :

la mémoire F663H contient la valeur 4
la valeur réelle se trouve de F7F6H à F7F9H

— S'il s'agit d'un réel en double précision

la mémoire F663H contient la valeur 8
la valeur réelle se trouve de F7F6H à F7FDH

— S'il s'agit d'une chaîne de caractères

la mémoire F663H contient 3

L'adresse du descripteur de la chaîne se trouve en F7F8H, F7F9H.

La valeur calculée par le programme en langage machine peut être passée au programme Basic de la même façon aux mêmes adresses.

*** VARPTR (<nom de variable>)**
VARPTR (<numéro de fichier>) **voir paragraphe 2.1.2**

2.2.2 Les fonctions de calcul

Pour toutes les fonctions mathématiques, le calcul se fait en double précision, quel que soit le type de l'argument employé.

*** ABS (X)**

Cette fonction donne la valeur absolue de x.

*** ATN (X)**

Cette fonction calcule l'angle en radians dont la tangente est x. Le résultat est compris entre $-\pi/_2$ et $\pi/_2$.

*** COS (X)**

Cette fonction donne le cosinus de l'angle x exprimée en radians.

*** EXP (X)**

Cette fonction calcule l'enponentielle de x en base e (e ↑ x) ; x doit être inférieur à 146.

*** LOG (X)**

Cette fonction donne le logarithme népérien de x ; x doit être supérieur à Ø.

*** SGN (X)**

Cette fonction donne − 1 si X est négatif
1 si X est positif
Ø si X est nul

*** SIN (X)**

Cette fonction calcule le sinus de l'angle x exprimé en radians.

*** SQR (X)**

Cette fonction calcule la racine carrée de x avec x supérieur ou égal à Ø.

*** TAN (X)**

Cette fonction donne la tangente de l'angle x exprimé en radians.

2.2.3 Traitement des chaînes de caractères

*** ASC (X$)**

Cette fonction donne le code ASCII du premier caractère de la chaîne x$. Si x$ est une chaîne vide, une erreur est provoquée.

*** CHR$ (X)**

Cette fonction renvoie le caractère correspondant au code ASCII x.

— les caractères imprimables ont un code ASCII compris entre 32 et 255.

— les caractères de contrôle ont un code ASCII inférieur à 32

— l'affichage de certains caractères graphiques dont les codes sont compris entre 65 et 95 peut être obtenu à la place des

caractères alphanumériques ayant les même codes. Il suffit pour cela d'utiliser la séquence d'échappement PRINT CHR$ (1).

Exemple

```
PRINT CHR$ (1) ; CHR$ (7Ø)
```

*** INSTR ([I,]X$, Y$)**

Cette fonction recherche la première apparition de la chaîne Y$ dans la chaîne X$, à partir du Iième caractère de X$ (le premier par défaut). La valeur fournie par INSTR est Ø si X$ n'est pas trouvée.

I doit être un nombre entier compris entre Ø et 255.

Si I est supérieur à la longueur de X$ ou si X$ est vide ou si X$ et Y$ sont vides alors INSTR renvoie Ø.

Si seule la chaîne Y$ est vide, INSTR donne 1, ou I si I est spécifié.

*** LEFT$ (X$, I)**

Cette fonction recopie dans la chaîne résultat la sous-chaîne de X$ constituée par les I premiers caractères de X$ (en commençant par la gauche).

I doit être un nombre entier compris entre Ø et 255.

Si I est supérieur à la longueur de X$, la chaîne résultat est X$.

Si I est nul, la chaîne résultat est vide.

*** LEN (X$)**

Cette fonction calcule la longueur de la chaîne X$. Tous les caractères, imprimables ou non, sont pris en compte.

*** MID$ (X$, I[, J])**

Cette fonction renvoie une chaîne de J caractères extraits de la chaîne X$ à partir de la I ième position.

I et J doivent être des entiers compris entre 1 et 255.

Si J n'est pas précisé ou s'il existe moins de J caractères dans X$ à partir de la I ième position, tous les caractères à droite du I ième élément constituent la sous-chaîne extraite.

Si I est supérieur à LEN (X$), la chaîne extraite est vide.

*** RIGHT$ (X$,I)**

RIGHT$ renvoie une chaîne constituée par les I derniers caractères de X$.

I doit être un entier compris entre Ø et 255.

Si I est nul, la chaîne résultat est vide.

Si I est supérieur à LEN (X$), toute la chaîne X$ est prise.

*** SPACE$ (X)**

Cette fonction donne une chaîne composée de X espaces ; X doit être un entier compris entre Ø et 255.

*** STR$ (X)**

Voir paragraphe 1.5.10

*** STRING$ (I, X$)**
STRING$ (I,X)

Cette fonction renvoie une chaîne contenant I fois le premier caractère de X$ ou I fois le caractère de code ASCII X.

*** VAL (X$)**

Voir paragraphe 1.5.10

2.3 CONTROLE DE L'ECRAN

Nous allons étudier dans ce paragraphe les instructions et fonctions suivantes :

KEY ON/OFF	WIDTH
KEY LIST	COLOR
CSRLIN	VDP
POS	
LOCATE	

Les instructions graphiques sont étudiées dans le chapitre 3 de ce livre.

*** KEY ON/OFF**

A la mise sous tension, la 24° ligne de l'écran est réservée pour l'affichage des chaînes correspondant aux touches de fonctions programmables.

KEY OFF supprime cet affichage et libère la ligne
KEY ON rétablit cet affichage.

*** KEY LIST**

Cette commande permet d'obtenir sur l'écran la liste des chaînes de caractères associées aux dix touches de fonction.

Exemple

```
KEY LIST
COLOR
AUTO
GOTO
LIST
RUN
COLOR 15, 4, 7
CLOAD«
CONT
LIST.
RUN
OK
```

*** CSRLIN**

Cette fonction renvoie la position verticale du curseur (entre Ø et 23).

*** POS (I)**

Cette fonction donne la position horizontale du curseur (entre Ø et 39) en mode texte 40 colonnes ; I est un argument fictif que peut prendre n'importe quelle valeur.

*** LOCATE [<x>] [,<y>] [,<curseur>]**

Cette commande permet de placer le curseur au point de coordonnées (x,y) en mode texte ($0 \leqslant x \leqslant 39$, $0 \leqslant y \leqslant 39$)

<curseur> valide ou non l'affichage du curseur clignotant
<curseur> = 1 curseur visible
<curseur> = 0 curseur invisible
<curseur> omis on reste dans le mode précédent.

Le coin supérieur gauche de l'écran a pour coordonnées (Ø,Ø). Des valeurs de X et Y n'appartenant pas respectivement à [Ø,39] et [Ø,23] provoquent une erreur.

Exemples

```
LOCATE 1Ø,1Ø,1
LOCATE, 5,Ø
LOCATE 2,,1
```

*** WIDTH <nombre de colonnes>**

Cette instruction permet de préciser le nombre de colonnes sur l'écran en mode texte ; <nombre de colonnes> doit être compris entre 1 et 4Ø en mode 4Ø × 24, et entre 1 et 32 en mode 32 × 24.

*** COLOR [<couleur texte>] [,<couleur fond>] [,<couleur bord>]**

L'instruction COLOR permet de préciser les couleurs des caractères, du fond de l'écran et de son bord. A la mise sous tension, COLOR 15,4,7 est pris par défaut.

Les couleurs possibles sont :

Ø transparent	8 rouge moyen
1 noir	9 rouge clair
2 vert moyen	1Ø jaune foncé
3 vert clair	11 jaune clair
4 bleu foncé	12 vert foncé
5 bleu clair	13 magenta
6 rouge foncé	14 gris
7 cyan	15 blanc

Pour plus de détails, reportez-vous au chapitre concernant les instructions graphiques.

*** VDP (X)**

VDP permet d'accéder directement aux registres du circuit contrôlant les signaux vidéos. Pour des valeurs entières comprises entre Ø et 7, le programmeur a accès en écriture à huit registres; VDP (8) renvoie un octet donnant l'état (registre de statut) du processeur graphique.

Cette fonction est à utiliser avec précaution car ses effets peuvent être... étonnants.

2.4 LE MAGNETOPHONE

Le MSX-Basic dispose d'instructions spéciales pour la sauvegarde et le chargement de programme sur une mémoire de masse constituée par une cassette magnétique.

Ces instructions sont :

MOTOR ON/OFF	CSAVE
SAVE	CLOAD
LOAD	CLOAD?
BSAVE	MERGE
BLOAD	

*** MOTOR ON/OFF**

Cette instruction permet de commander le moteur du magnétophone à cassettes si celui-ci possède une prise pour télécommande.

MOTOR ON met en marche le moteur
MOTOR OFF arrête le moteur
MOTOR arrête le moteur s'il était en marche et le lance s'il était à l'arrêt.

*** SAVE «<type de périphérique> <nom de fichier>»**

Cette commande permet de sauvegarder un programme en mémoire utilisateur dans la mémoire d'un périphérique.

- <type de périphérique> indique sur quel périphérique s'effectue la sauvegarde, il peut s'agir de :

  — cassette audio CAS :
  — écran basse résolution CRT :

— écran graphique GRP :
— imprimante LPT :

• <nom de fichier> est constitué par une suite de six caractères au maximum, suivie d'un point et d'un suffixe optionnel indiquant s'il s'agit d'un fichier Basic (BAS), de données (DAT) ou binaire (BIN).

*** LOAD « <type de périphérique> <nom de fichier> » [,R]**

Cette commande est complémentaire de la précédente puisqu'elle permet de charger à partir d'un périphérique un programme en mémoire centrale.

Pour les arguments, se référer à la commande SAVE.

Quelques remarques :

— LOAD ferme tous les fichiers ouverts en cours (équivalent de CLOSE) et détruit le programme présent en mémoire utilisateur.
— Si l'option R est utilisée, le programme chargé est immédiatement exécuté.
— Si <nom de fichier> est omis, le premier programme rencontré est chargé.

*** BSAVE « <type de périphérique> <nom de fichier> », <adresse de début> <adresse de fin> [,<adresse d'exécution>]**

C'est une commande de sauvegarde sur cassette d'une zone mémoire

<type de périphérique> « CAS : » pour le magnétophone cassette

<nom de fichier> par exemple « JEU »

<adresse de début>, <adresse de fin> sont les arguments qui indiquent les adresses du début et de fin de la zone mémoire à sauvegarder.

<adresse d'exécution> indique l'adresse du début du programme.

Si <adresse d'exécution> est omis, l'adresse de début du programme devient l'adresse d'exécution.

Exemples

```
BSAVE «CAS : JEU», &HØØØØ, &HAFFF, &HEØØØ
BSAVE «CAS : TEST.BIN», &HØØØØ, &HAFFF
```

*** BLOAD «<type de périphérique> <nom de fichier>» [,R] [offset]**

Cette commande permet le chargement à partir d'un périphérique d'un programme en langage machine.

Avec <type de périphérique> « CAS : » pour le magnétophone.

Si l'option R est notifiée, le programme est immédiatement exécuté après son chargement à partir de l'adresse spécifiée lors de sa sauvegarde.

Si l'argument <offset> est indiqué, toutes les adresses sont translatées de la valeur de <offset>.

Si <nom de fichier> est omis, le premier programme rencontré est chargé.

*** CSAVE «<nom de fichier>» [,<vitesse de transmission>]**

La commande exécute la sauvegarde d'un programme Basic sur une cassette audio.

Cette commande, un peu particulière, effectue la sauvegarde d'un fichier dans un format binaire compressé ; en effet, un fichier ASCII prend une place importante en mémoire ou sur une cassette, d'où l'intérêt de cette commande qui minimise l'encombrement d'un programme au moment de sa sauvegarde.

<vitesse de transmission> peut prendre la valeur 1 ou 2 : 1 pour 1 200 bauds et 2 pour 2 400 bauds.

*** CLOAD [«<nom de fichier>»]**

C'est la commande complémentaire de CSAVE.

Si <nom de fichier> est omis, le premier programme présent sur la cassette est chargé. La vitesse de transmission est déterminée automatiquement.

*** CLOAD? [«<nom de fichier>»]**

Cette commande permet de vérifier l'identité entre le programme chargé en mémoire et le programme source écrit sur cassette.

Si <nom de fichier> est omis, la comparaison se fait avec le premier programme rencontré sur cassette.

CLOAD?, que l'on rencontre souvent dans d'autres Basic sous le nom de VERIFY, est une instruction très précieuse pour s'assurer de la qualité d'une sauvegarde sur cassette. La fiabilité des interfaces cassette n'est malheureusement pas souvent le point fort des micro-ordinateurs...

*** MERGE «<type de périphérique> [<nom de fichier>]»**

Cette commande permet d'additionner un programme sur cassette à un programme en mémoire.

<type de périphérique> CAS : pour magnétophone.

Si <nom de fichier> est omis, le premier fichier ASCII rencontré est celui utilisé par la commande.

Si des numéros de ligne du fichier à recopier sont présents dans le programme en mémoire, les lignes du fichier remplacent les lignes correspondantes en mémoire.

2.5 LES MANETTES DE JEUX

La norme MSX prévoit que les machines doivent être équipées d'un port d'entrée sur lequel on puisse brancher deux manettes de jeux. Le Basic-MSX possède plusieurs instructions pour gérer ces manettes ou les périphériques raccordés à ce port :

ON STRIG GOSUB	STICK
STRIG (N) ON/OFF/STOP	PDL
STRIG	PAD

*** ON STRIG GOSUB [<numéro 1 de ligne>] [,<num 2>]...[,<num 5>]**

Le principe est le même que pour ON INTERVAL GOSUB, ON STOP GOSUB... L'action sur le bouton-poussoir d'une manette de jeu génère une interruption. Si cette interruption n'est pas masquée, il y a déroutement vers le sous-programme correspondant au périphérique qui a généré l'interruption.

<num 1> correspond à la barre du clavier,
<num 2> et <num 4> correspondent au bouton de la manette 1,
<num 3> et <num 5> correspondent au bouton de la manette 2.

Exemple

ON STRIG GOSUB, 1ØØ,,1ØØ ne prend en compte que le bouton de la manette 1.

Chaque fois qu'un branchement est effectué, un STRIG (N) STOP est automatiquement fait. Au retour du sous-programme, un STRIG (N) ON est fait.

*** STRIG (N) ON/OFF/STOP**

Cette instruction valide, masque ou inhibe l'interruption générée par la touche correspondant à N.

N = Ø correspond à la barre du clavier
N = 1 et N = 3 correspondent au bouton-poussoir de la manette 1
N = 2 et N = 4 correspondent au bouton-poussoir de la manette 2

*** STRIG (N)**

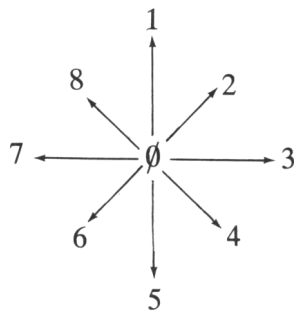
Cette fonction donne le statut de la touche correspondant à N (cf. ci-dessous). Si la touche est pressée, STRIG(N) vaut 1 sinon STRIG(N) vaut Ø.

*** STICK(N)**

Cette fonction donne la direction du manche à balai pour la manette de jeu correspondant à N.

— Si N = Ø ce sont les touches de déplacement du curseur qui sont sélectionnées
— Si N = 1 c'est la manette 1 qui est sélectionnée
— Si N = 2 la manette 2 est sélectionnée.

La valeur renvoyée par STICK en fonction de l'action sur la manette est montrée par la figure suivante.



Pour obtenir un déplacement en diagonale avec les touches de déplacement du curseur, il faut appuyer simultanément sur deux touches.

*** PDL(N)**

Cette fonction donne une valeur dépendant de l'accessoire raccordé sur le port d'entrée-sortie. Si N est pair, c'est le port correspondant à la manette 2 qui est sélectionné, alors que si N est impair, c'est le port correspondant à la manette 1.

*** PAD(N)**

Cette fonction renvoie le statut d'un *paddle.*

Un exemple complet d'application des instructions décrites ci-dessus est donné dans le chapitre 3

2.6 LE CLAVIER

Le standard MSX prévoit que toutes les machines répondant à la norme aient un clavier possédant certaines caractéristiques. En particulier les claviers MSX doivent tous posséder 73 touches dont cinq permettant d'accéder à dix fonctions préprogrammées.

Nous avons déjà vu que les fonctions choisies par le système à la mise sous tension sont :

F1 COLOR [B]
F2 AUTO [B]
F3 GOTO [B]
F4 LIST [B]
F5 RUN [RC]
F6 COLOR 15,4,7[RC]
F7 CLOAD»
F8 CONT [RC]
F9 LIST. [RC] [U] [U]
F1Ø[CLS] RUN [RC]

[B] espace
[RC] retour chariot
[CLS] effacement de l'écran
[U] curseur vers le haut

Le MSX-Basic dispose d'instructions particulieres pour gérer ces touches. Il s'agit de :

KEY
ON KEY GOSUB
KEY ON/OFF/STOP

Nous avons déjà étudié dans des paragraphes précédents : KEY ON/OFF et KEY LIST

*** KEY <numéro de touche>, <chaîne de caractère>**

L'instruction KEY affecte la touche dont le numéro est spécifié la chaîne de caractère.

<numéro de touche> doit être un nombre entier compris entre 1 et 1Ø.

<chaîne de caractères> ne doit pas comporter plus de quinze caractères.

En pressant une touche (il faut aussi presser SHIFT pour les fonctions 6 à 1Ø), on obtient sur l'écran l'affichage de la chaîne de caractères à la touche pressée. Si la chaîne de caractères correspond à une instruction exécutable du Basic et qu'elle est terminée par un caractère de contrôle RC (code ASCII 13), la commande s'exécute quand on presse la touche.

Exemple

KEY 2, «CLS» + CHR$ (13) permettra d'obtenir l'effacement de l'écran en appuyant sur la touche 2.

*** ON KEY GOSUB [<numéro de ligne 1>] [,<num 2>]... [,<num 1Ø>]**

La «philosophie» de cette instruction est la même que celle de ON STOP GOSUB, ON INTERVAL GOSUB... Le déroutement a lieu si l'interruption générée par la pression d'une touche de fonction est autorisée par KEY(N)ON où N est le numéro de la touche correspondante.

La liste des numéros donnés en argument de ON KEY GOSUB permet un branchement vers dix sous-programmes différents.

Exemple

ON KEY GOSUB 1Ø,,, 2Ø, 3Ø indique qu'il y aura branchement automatique à un sous-programme pour les touches 1, 4, et 5.

Chaque fois qu'un déroutement est effectué, un KEY(N) OFF est effectué ; au retour du sous-programme un KEY(N) ON est fait automatiquement sauf mention explicite d'un KEY(N) OFF dans le sous-programme.

De même que pour les autres instructions du même type, les branchements ne sont possibles qu'en mode programme.

* KEY ON/OFF/STOP

Cette instruction permet de valider, d'inhiber ou de mémoriser une demande d'interruption provoqué par la pression d'une touche de fonction.

Exemple

```
 1Ø FOR I = 1 TO 5
 2Ø KEY (I) ON
 3Ø NEXT I
 4Ø ON KEY GOSUB15Ø,2ØØ,25Ø,3ØØ,35Ø
 5Ø CLS : A = 1 : A$ = «TOUCHE DE FONCTION NO»
 6Ø FOR I = 1 TO 1ØØ
 7Ø IFA = Ø THEN END
 8Ø PRINT I;
 9Ø FOR I = Ø TO 5Ø : NEXT T : REM TEMPO
1ØØ PRINT CHR$ (13);
11Ø NEXT I
12Ø PRINT
13Ø END
15Ø A = Ø : PRINT
16Ø PRINT A$; «1»
17Ø RETURN
2ØØ A = Ø : PRINT
21Ø PRINT A$; «2»
22Ø RETURN
25Ø A = Ø : PRINT
26Ø PRINT A$; «3»
```

```
27Ø RETURN
3ØØ A = Ø : PRINT
31Ø PRINT A$ ; «4»
32Ø RETURN
35Ø A = Ø : PRINT
36Ø PRINT A$ ; «5»
37Ø RETURN
```

Après la partie d'initialisation des interruptions sur cinq touches de fonction (lignes1Ø à 4Ø), le programme principal fait tourner un compteur sur l'écran (lignes 6Ø à 13Ø). A tout moment le programme peut être interrompu en pressant simplement l'une des cinq touches de fonction. Le numéro de la touche pressée est alors affiché.

CHAPITRE 3

Commandes graphiques et sonores

Le Basic MSX apporte, au niveau des commandes graphiques et sonores, de nombreuses améliorations par rapport à beaucoup d'autres interpréteurs. Il est donc normal de leur accorder un chapitre entier dans cet ouvrage d'introduction.

En effet, le Basic MSX a été plus spécialement conçu pour élaborer des programmes de jeux, comme le montre la part très importante de ceux-ci dans la bibliothèque de programmes existant au Japon au moment de la rédaction de ce livre.

Les possibilités graphiques ont été élargies par l'adjonction de commandes de tracé comme CIRCLE ou DRAW, de commandes de remplissage de formes comme PAINT et de commandes de manipulation de figurines ou lutins comme PUT SPRITE ou ON SPRITE GOSUB. Toutes ces instructions facilitent grandement le travail du programmeur et surtout minimisent leur propre temps d'exécution dans un programme.

Malgré la présence de deux instructions seulement (PLAY et

SOUND), les possibilités sonores de Basic MSX sont très étendues grâce à la présence d'un circuit générateur de son très performant.

3.1 COMMANDES GRAPHIQUES

Avant de commencer l'étude des commandes graphiques, il est nécessaire de connaître l'organisation de la mémoire vidéo afin d'élaborer les valeurs à initialiser en cas d'accès direct à cette mémoire (instructions VPEEK, VPOKE).

3.1.1 Organisation de la mémoire vidéo en haute résolution

L'écran graphique en haute résolution se compose de 192 lignes et 256 pixels. Le système Basic MSX dispose de 16 couleurs.

Un rapide calcul montre qu'il faut 49 152 bits pour allumer ou éteindre tous les pixels, soit 6 144 octets ou 6K octets. Pour les allumer en 16 couleurs, il faut multiplier par 4 ce nombre, d'où 24K octets pour accéder à tous les pixels avec 16 couleurs.

Or, la zone mémoire réservée à la visulatisation est au maximum de 16K octets. Il existe donc une limitation au niveau des couleurs pour un point élémentaire.

En effet, 8 pixels consécutifs d'une même ligne sont codés sur un octet. Pour chaque point codé, il n'existe que deux couleurs possibles par octet, puisque chacun de ses bits ne peut prendre que deux valeurs 0 ou 1. Pour coder ces deux couleurs, un octet est nécessaire pour chaque groupe de 8 pixels. Donc le système comporte deux groupes d'octets :

- De l'adresse vidéo 0 à 6143 : les octets codent les pixels. L'ordre des octets sur l'écran est schématisé par la figure suivante :

M−8	Adr. Vidéo M	M+8
M−7	M+1	M+9
M−6	M+2	M+1Ø
M−5	M+3	M+11
M−4	M+4	M+12
M−3	M+5	M+13
M−2	M+6	M+14
M−1	M+7	M+15

• De l'adresse vidéo 8 192 à 14 335 : associés chacun à un octet du groupe précédent, ces octets codent les deux couleurs possibles des 8 pixels correspondants.

Le codage des couleurs se fait de la manière suivante :

7	6	5	4	3	2	1	0
Couleur des bits à 1				Couleur des bits à 0			

Lors de l'exécution de l'instruction SCREEN, les mémoires du premier groupe sont mises à zéro, alors que celles du deuxième sont initialisées suivant la dernière commande COLOR exécutée ; c'est-à-dire que la couleur des bits à 0 est la dernière couleur de fond demandée et que la couleur des bits à 1 est la dernière couleur des bords.

Grâce aux instructions VPEEK et VPOKE et aux informations données dans cette partie, le programmeur peut dessiner des figures ou créer des routines de tracé en agissant directement sur la mémoire vidéo en haute résolution.

L'adresse que contient les instructions VPEEK et VPOKE peut prendre une valeur de 0 à 16385 (16 K-octets). Il reste 4 K-octets (de 6144 à 8191 et de 14336 à 16385) qui sont utilisés pour la gestion des instructions de tracé tel que DRAW ou des instructions de manipulation des lutins ou sprites. Par exemple, les lutins qui sont du type chaînes de caractères sont mémorisés à partir de l'adresse 14336.

3.1.2 Instructions de contrôle de l'écran

Lorsque un micro-ordinateur est mis sous tension, il se trouve obligatoirement en mode TEXTE : l'affichage n'est autorisé qu'aux caractères alphanumériques, spéciaux ou semi-graphiques. La mémorisation de la page d'écran s'effectue au niveau du caractère et non du pixel : chaque caractère est alors traité lors de son affichage par un générateur de caractères qui contient tous les motifs de visualisation.

Par contre, pour un affichage graphique, qu'il soit en basse ou haute résolution, la mémoire contient l'état de chaque pixel, ou de

chaque groupe de pixel. Le générateur est alors inutile. Il est donc nécessaire de posséder une commande de commutation du mode texte en mode graphique et vice versa.

Dans le cas du Basic MSX et comme pour de nombreux autres interpréteurs, l'instruction SCREEN est chargée de cette fonction de commande entre les modes d'affichage.

Commençons l'étude de cette commande. La syntaxe en est la suivante :

SCREEN **[<mode>][,<taille des lutins>]**
[,<validation du déclic des touches>]
[,<vitesse de transmission cassette>]
[,<imprimante MSX>]

Cette instruction ne fait pas que sélectionner le mode de visualisation, mais permet aussi de choisir la taille des lutins affichés, de valider ou d'inhiber le déclic des touches, de définir la vitesse de transmission avec le magnétophone à cassettes et d'indiquer la présence d'une imprimante aux normes MSX.

a) Sélection du mode d'affichage : Le Basic MSX en possède 4

— Le mode texte en 24 lignes de 40 caractères.
<mode>=0

— Le mode texte en 24 lignes de 32 caractères.
<mode>=1

— Le mode graphique en haute résolution.
<mode>=2

L'étude de ce cas a été effectuée dans la partie précédente (3.1.1). Les caractéristiques importantes sont :

- 256 × 192 pixels
- 16 couleurs mais seulement 2 par groupes de 8 pixels.

— Le mode graphique en basse résolution.
<mode>=3

Les deux grandes différences avec le mode en haute résolution sont d'abord la résolution puisque dans ce cas, nous ne disposons que de 64 × 48 points et ensuite les possibilités en couleur des points.

En effet, ce mode s'appelle aussi le mode multi-couleur : cela signifie que les 16 couleurs peuvent être attribuées à chaque point sans aucune limitation. Ceci est rendu possible par le fait que chaque octet auquel étaient attribuées deux couleurs ne représente plus qu'une partie de deux points.

b) Choix de la taille des lutins.

Les lutins se composent de deux sortes de matrices de points : soit 8 par 8, soit 16 par 16. De plus, l'interpréteur Basic comprend un mode loupe pour les lutins : c'est-à-dire que la surface des lutins est multiplié par 4. Nous avons donc à notre disposition 4 tailles de figurines :

— 8 × 8 points :
<taille des lutins>=0

— 8 × 8 points en mode loupe :
<taille des lutins>=1

— 16 × 16 points :
<taille des lutins>=2

— 16 × 16 points en mode loupe :
<taille des lutins>=3

Les valeurs 0 et 1 n'autorisent que l'affichage des lutins de 8×8 pixels alors que les valeurs 2 et 3 valident la visualisation des deux tailles de figurines.

La commande SCREEN doit être exécutée avec les instructions d'assignation des lutins (SPRITE$) parce qu'elle initialise à 0 les contenus des figurines.

c) Validation du déclic des touches

Ce paramètre valide ou inhibe le déclic des touches au moment de leur frappe par initialisation à 1 ou à 0 respectivement.

d) Choix de la vitesse de transmission

Deux vitesses de transmission avec le magnétophone à cassettes sont possibles. Ce choix peut être effectué au moment de l'instruction SCREEN, mais aussi au moment même de la sauvegarde

par la commande CSAVE. Dans les deux cas, la valeur 1 permet une écriture à 1 200 bands et la valeur 2 à 2 400 bands.

Lors de la lecture de programmes sur cassette, la sélection de la vitesse de transmission est inutile, car celle-ci est déterminée automatiquement.

e) Imprimante aux normes MSX

Si l'imprimante connectée au système possède les symboles graphiques du Basic MSX, ce paramètre doit être mis à 0 et à 1 dans le cas contraire.

Les instructions graphiques telles que PUT SPRITE ou LINE provoquent, en mode texte, l'affichage du message *Illegal function call :* appel illégal de fonctions. Par contre, l'exécution de l'instruction INPUT force le système en mode texte. Il en est de même pour les instructions de fin de programme END ou STOP. Par conséquent si le programmeur désire rester en affichage graphique à la fin d'un programme, il doit le faire boucler sur une instruction GOTO.

Au niveau de la commande des couleurs par l'instruction COLOR, la visualisation des modes graphiques fait apparaître la couleur de bord qui est le troisième paramètre de l'instruction :

```
COLOR [<couleur de caractères>]
      [,<couleur de fond>]
      [,<couleur de bord>]
```

3.1.3 Instructions de visualisation

Ce type d'instruction agit soit directement sur la mémoire d'écran par un adressage physique de celui-ci (comme VPEEK et VPOKE) soit indirectement par les coordonnées des points à afficher (comme PSET, PRESET et POINT).

*** CLS**

Cette instruction effectue une remise à zéro de la mémoire d'écran quel que soit son mode d'utilisation.

*** VPEEK (<adresse en mémoire d'écran>)**

Cette fonction retourne la valeur contenue à l'adresse spécifiée

dans la mémoire d'écran. Cette adresse doit être comprise dans l'intervalle [0 ... 16383]

Exemple : A = VPEEK (255)

*** VPOKE <adresse en mémoire vidéo>,<valeur>**

Cette instruction est complémentaire de la fonction, VPEEK. En effet, elle permet d'écrire une valeur en mémoire d'écran à l'adresse spécifiée (entre 0 et 16383).

L'emploi des deux instructions précédentes s'effectue en fonction des remarques faites sur l'organisation de la mémoire vidéo. voir paragraphe 3.1.1.

*** PSET <X,Y> [,<couleur>]**
PRESET <X,Y> [,<couleur>]

Ces commandes attribuent une couleur à un point quelconque de l'écran graphique. Deux cas sont à étudier suivant que le mode graphique est en basse ou haute résolution :

— En basse résolution, aucune limitation sur les couleurs des points n'existe. Par conséquent, le programmeur est libre d'attribuer la couleur qu'il désire à un point quelconque.

— En haute résolution, seules deux couleurs sont possibles par groupe de 8 points. Donc, la couleur du dernier point qui est initialisé devient celle du point du même octet dont les bits correspondants sont à 1. Ce changement de couleur est très gênant aux intersections de tracés de différentes teintes.

Si le paramètre <couleur> n'est pas spécifié dans la commande PSET, la couleur courante (c'est-à-dire la dernière spécifiée dans une commande graphique) est utilisée.

Dans le cas de l'instruction PRESET, la couleur utilisée est alors la couleur de fond de la dernière commande de COLOR exécutée.

*** POINT (<X,Y>)**

Cette fonction retourne la couleur du pixel spécifié par les coordonnées X,Y

3.1.4 Instructions de tracé

Connaissant les instructions permettant d'accéder à un point de l'écran, soit directement par adressage de la mémoire vidéo, soit indirectement par les coordonnées, il est facile d'imaginer la somme de commandes de ce type nécessaires aux tracés d'une droite, ou d'une figure géométrique telle qu'un carré ou un cercle.

Pour faciliter de tels tracés, le Basic MSX se devait d'intégrer les commandes de tracé de droites, de carrés ou de cercles.

* **LINE [(<coordonnées de début>)]−(<coordonnées de fin>) [,<color>][,<B/BF>]**

Les coordonnées de début et de fin peuvent être spécifiées sous deux formes :

— en valeurs absolues : <X,Y>

— en valeurs relatives aux coordonnées précédentes : <STEP (offset sur X, offset sur Y)>

(0,0) correspond au coin supérieur gauche.

Il faut remarquer que si les valeurs spécifiées sont extérieures aux intervalles [0,255] pour X et [0,191] pour Y, elles sont remplacées par la valeur la plus proche d'elles appartenant aux intervalles. Par exemple, 0 remplace toutes les coordonnées négatives.

Le programmeur peut aussi spécifier la couleur dans laquelle le tracé doit s'effectuer. Par défaut, la couleur définie par la dernière instruction COLOR est choisie.

Avec ces trois paramètres (ou deux puisque les premières coordonnées ne sont pas obligatoires), cette fonction trace une ligne entre les points.

Mais il est possible de tracer un carré, ou plutôt un rectangle car un pixel n'est pas forcément un carré : cela dépend du réglage du moniteur vidéo ou du poste de télévision.

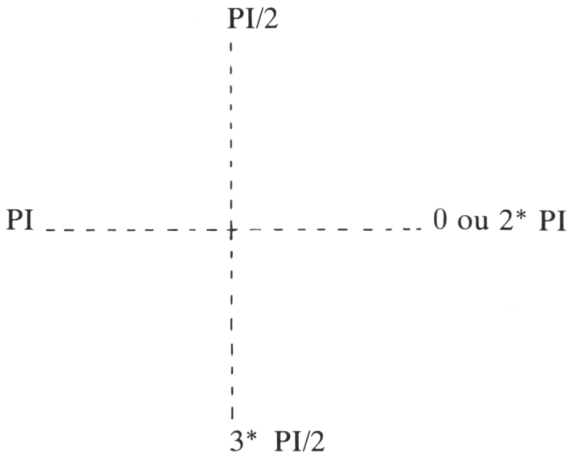
En ajoutant l'option B, la figure géométrique est vide, et avec l'option BF, elle est pleine.

*** CIRCLE (<coordonnées du centre>), <rayon>[,<couleur>]**
[,<angle de début>][,<angle de fin>]
[,<rapport des axes>]

Cette instruction permet de tracer un cercle à partir des coordonnées du centre (absolues ou relatives) et de la longueur (entière) du rayon.

La couleur du tracé est définie soit par celle qui est indiquée dans la commande, soit par la couleur de caractère spécifiée par la dernière instruction COLOR.

Il est possible de ne dessiner qu'une partie du cercle en spécifiant, dans l'instruction, l'angle de début et l'angle de fin du tracé. Les angles sont repérés comme l'indique la figure suivante :



Si le micro-ordinateur exécute l'instruction : CIRCLE (128,96),60,11 la forme obtenue n'est pas un cercle mais une ellipse pour la même raison qui fait que le tracé d'un carré par LINE avec l'option B, donne un rectangle.

Le paramètre <rapport des axes> permet de corriger cette ellipse pour qu'elle devienne un cercle ou alors de tracer différents types d'ellipses.

Exemple

```
CIRCLE(128,96),80,13,,,2
CIRCLE(128,96),80,13,,,0.25
```

Ces instructions doivent être placées dans un programme bouclant sur lui-même pour que le tracé reste visible.

*** PAINT <coordonnées de départ>[,<couleur de remplissage>] [,<couleur limite>]**

Cette commande effectue le remplissage de figures graphiques dans une couleur spécifique : <couleur de remplissage>. Cette fonction s'effectue jusqu'aux limites de la figure qui doivent être définies par une seule couleur <couleur limite>.

Les coordonnées de départ indiquent à partir de quel endroit s'effectue le remplissage : si le point se situe à l'intérieur, toute la figure est peinte. Dans le cas contraire, le remplissage s'exécute à l'extérieur.

Les trois instructions décrites précédemment sont puissantes, mais limitent beaucoup la complexité des figures géométriques, à moins de travailler point par point. Cela nécessite alors une conception lourde et des longues mises au point. Le Basic MSX se devait de libérer le programmeur pour le tracé de figures complexes : d'où l'instruction DRAW associée à un macro-langage graphique.

*** DRAW <chaîne de caractère>**

Les commandes graphiques sont contenues dans la chaîne alphanumérique que l'interpréteur Basic analyse caractère par caractère. L'allumage des pixels s'effectue à chaque déplacement commandé par le programmeur, sauf si l'option de non-affichage est spécifiée.

1) Commandes de déplacement relatif.

U<n> : déplacement vers le haut (Up)
D<n> : déplacement vers le bas (Down)
L<n> : déplacement vers la gauche (Left)
R<n> : déplacement vers la droite (Right)
E<n> : déplacement en diagonale vers le haut et à droite

F<n> : déplacement en diagonale vers le bas et à droite
G<n> : déplacement en diagonale vers le bas et à gauche
H<n> : déplacement en diagonale vers le haut et à gauche

n indique le nombre de points à parcourir dans le cas où le facteur d'échelle est 1 (défini par S)

2) Commande de positionnement

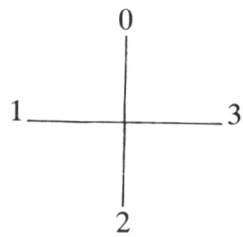
M<x,y> exécute un déplacement absolu ou relatif.
— en relatif, les valeurs de x et y doivent être précédées par + ou −
— en absolu, les valeurs de x et y sont positives sans signe.

La droite joignant les 2 points est tracée dans la couleur courante.

3) Commande de déplacement angulaire.

A<n> : n est compris entre 0 et 3.

0 pour 0 degrés
1 pour 90 degrés
2 pour 180 degrés
3 pour 270 degrés



4) Deux commandes se placent comme préfixe de n'importe quelle autre macro-instruction.

— B : Le déplacement effectué est celui qui est spécifié par le caractère suivant mais aucun affichage ne s'exécute.

— N : Le déplacement et l'affichage s'effectuent normalement mais il y a retour à la position initiale.

5) Commande de couleur

C<n> : initialisation de la couleur du tracé.

6) Facteur d'échelle

S<n> : Cette macro-commande fixe le facteur d'échelle de déplacement.
n doit être compris entre 0 et 255, mais le facteur d'échelle est divisé par 4. Cette valeur multiplie chaque paramètre de toutes les instructions de déplacement U, D, L, R, E, F, G, H et N en mode relatif).

Par défaut, le facteur d'échelle est égal à 1 (c'est-à-dire n = 4).

7) Exécution de sous-chaîne de caractère

Une même séquence de caractères peut se retrouver souvent dans les exécutions DRAW : Il est donc préférable de lui assigner un nom de variable. Il faut donc avoir une macro-commande d'exécution de sous-chaînes de caractères :

X<variable-chaîne> ; (Ne pas oublier le ;)

Dans toutes ces macro-instructions, les paramètres x, y et n peuvent être des constantes entières ou des variables entières.

Voilà terminée la description d'une des commandes les plus puissantes du Basic MSX. Ses applications sont nombreuses ; nous pouvons par exemple imaginer un programme d'aide au dessin qui permettrait d'exécuter un dessin à partir des déplacements d'un point-curseur, en validant le tracé de droites entre des points-tests...

3.1.5 Génération et manipulation des lutins

Il est facile d'imaginer les problèmes introduits par la génération et surtout par la manipulation de figurines en point par point sur l'écran en haute résolution. Etant données les prétentions affichées par le Basic MSX au niveau du graphisme, il se devait d'intégrer des instructions facilitant la mémorisation, mais surtout les déplacements des lutins. De plus, ces commandes doivent être les plus rapides possibles pour éviter à l'utilisateur la séquence programmée d'opérations : effacement/apparition.

Nous allons donc commencer par l'instruction de définition des lutins : SPRITE$ (<numéro du lutin>).

En fait, il s'agit plus d'une variable que d'une instruction. Elle est

type chaîne de caractères : elle peut assigner ou être assignée par une variable de ce type. Quelle que soit la taille des figurines 8 par 8 ou 16 par 16, leur longueur est fixée à 32 octets. Le caractère ASCII 0 complète en cas de chaîne plus courte.

Le numéro de lutin fixé au moment de l'initialisation est celui qui sera utilisé pour effectuer les déplacements par l'instruction PUT SPRITE. Ce numéro doit être inférieur à 256 pour des figurines 8 par 8, et à 64 pour des lutins 16 par 16.

Exemples de définition des lutins.

Le but est de créer une chaîne de caractères à partir du dessin du lutin en fonction de la taille de figurine désirée. Le point de départ est la matrice de pixels soit 8 × 8 soit 16 × 16. L'introduction de ce tableau peut se faire par l'intermédiaire de l'instruction DATA :

```
DATA & B 0 0 0 1 0 0 0 0
DATA & B 0 0 1 1 1 0 0 0
DATA & B 0 0 0 1 0 0 0 0
DATA & B 0 0 1 1 1 0 0 0
DATA & B 0 1 0 1 0 1 0 0
DATA & B 0 0 0 1 0 0 0 0
DATA & B 0 0 1 0 1 0 0 0
DATA & B 0 1 0 0 0 1 0 0
```

Chaque ligne peut être convertie en une valeur hexadécimale ou décimale pour n'obtenir qu'une instruction DATA.

Exemple

```
DATA &H1Ø, &H38, &H1Ø, &H38, &H54, &H1Ø, &H28, &H44
```

L'avantage de la première solution est que les données sont beaucoup plus significatives que dans le deuxième cas.

Les fonctions VAL et CHR$ sont alors utilisées pour convertir ces données de caractères en valeurs numériques puis en caractères pour être concaténés en une chaîne. Celle-ci sera alors assignée à la variable SPRITE$. Ce qui donne pour la séquence de lecture des « DATA » :

```
S$=« »
FOR T=1 TO 8
READ A$
```

```
S$ = S$ + CHR$(VAL(A$))
NEXT T
SPRITE$(1)=S$
```

Pour des lutins de 16×16 pixels, il est beaucoup plus rapide de dessiner chaque ligne de dessin sous forme de deux nombres décimaux. En effet, la fonction CHR$ ne travaille que sur des nombres inférieurs à 256. Deux exemples de définition d'une figurine 16 par 16 sont donnés dans le programme de jeu, à la fin de cette partie réservée au graphisme.

Maintenant que les lutins sont initialisés, il faut pouvoir les placer et surtout les déplacer sur l'écran. Ces deux fonctions sont réalisées par la même instruction :

PUT SPRITE <numéro de plan>[,<coordonnées X,Y>]
[,<couleur du lutin>][,<numéro du lutin>

Avant de commencer la description des paramètres, il est nécessaire de faire une remarque touchant la mémorisation des lutins. Celle-ci ne s'effectue pas dans la mémoire d'écran : ils sont stockés dans la mémoire à l'adresse 1Ø336 jusqu'à 1536Ø (1 K-octets) pour être visualisés en superposition avec l'image graphique. Comme les lutins nécessitent 32 octets, seuls 32 lutins (1 Ø24 divisé par 32) peuvent apparaître en même temps sur l'écran, en superposition : il existe donc 32 plans de visualisation.

La présence de ces 32 plans explique qu'il n'existe pas, en cas de superposition des lutins, de conflit de couleurs. Cela est d'un intérêt non négligeable car la limitation en couleurs de l'image nuit à sa qualité. La superposition des couleurs se fait par priorité des plans : le plan Ø est le plan supérieur.

Les paramètres de l'instruction PUT SPRITE sont les suivants :

— <numéro de plan> : il indique dans quel plan placer le lutin spécifié par <numéro de lutin>. Un plan ne peut contenir qu'un seul lutin : l'insertion d'une autre figurine ou le déplacement du lutin sur un même plan efface sa position précédente.

— <coordonnées X,Y> : elles spécifient le lieu d'insertion du lutin : elles sont données soit en absolu soit relativement aux dernières coordonnées indiquées ; X doit être compris entre −32 et 255 et Y entre −32 et 191. Les valeurs négatives permettent de ne faire apparaître qu'une partie du lutin au bord de l'écran.

Deux valeurs particulières peuvent être attribuées à Y, ce qui provoque deux effets non moins particuliers :

- si Y=2Ø8, tous les plans inférieurs ou égaux en priorité (c'est-à-dire supérieurs ou égaux par leur numéro) sont inhibés de la visualisation.
- si Y=2Ø9, le lutin indiqué par l'instruction est inhibé pour l'affichage.

— <couleur de lutin> : elle spécifie la couleur de visualisation de la figurine.

— <numéro de lutin> : il indique quel lutin doit être visualisé sur le plan de l'instruction.

Grâce aux deux instructions précédentes, nous savons créer et déplacer jusqu'à 32 lutins sur une image graphique. Or, lorsqu'il y a collision entre deux d'entre eux dan un jeu vidéo, en général ils disparaissent. Du point de vue séquencement du programme, cette superposition doit donc être détectée Le Basic MSX intègre une instruction qui génère une interruption du même type que celle créée par les touches de fonction ou les boutons des joy-sticks. Cette instruction est **ON SPRITE GOSUB <numéro de ligne>,** à laquelle peut être associée l'instruction de validation **SPRITE ON/OFF/STOP**.

Ces deux commandes fonctionnent comme ON STRIG GOSUB ou ON INTERVAL GOSUB décrites précédemment. Il ne restera donc qu'à indiquer quelle configuration crée une interruption. Celle-ci est générée lorsque deux figurines se superposent au niveau du dessin même.

Nous disposons maintenant de toutes les informations et de toutes les instructions permettant la création et la manipulation des lutins. Un exemple de programmation de ces commandes est donné sous forme d'un jeu. Le but en est de faire entrer une balle rouge dans un cadre jaune par l'ouverture sans toucher les bords. Entre la position initiale de la balle et le cadre, apparaissent et disparaissent aléatoirement toutes les demi-secondes, des obstacles que le joueur doit éviter.

Ce programme ne présente pas de difficultés du point de vue du jeu mais il a le mérite d'utiliser de nombreuses commandes du Basic MSX et donc de donner un exemple de leur fonctionnement.

```
5     REM      JEU
1Ø    REM      INIT BALLE
2Ø    DATA     &B Ø Ø Ø Ø Ø Ø Ø Ø
3Ø    DATA     &B Ø Ø Ø Ø Ø Ø Ø Ø
4Ø    DATA     &B Ø Ø Ø Ø Ø Ø Ø Ø
5Ø    DATA     &B Ø Ø Ø 1 1 Ø Ø Ø
6Ø    DATA     &B Ø Ø Ø 1 1 Ø Ø Ø
7Ø    DATA     &B Ø Ø Ø Ø Ø Ø Ø Ø
8Ø    DATA     &B Ø Ø Ø Ø Ø Ø Ø Ø
9Ø    DATA     &B Ø Ø Ø Ø Ø Ø Ø Ø
1ØØ   S$=« »:REM LECTURE LUTIN N° 2
11Ø   FOR   T=1 TO 8
12Ø   READ  A$
13Ø   S$=S$+CHR$(VAL(A$))
14Ø   NEXT  T
15Ø   REM
16Ø   REM   INITIALISATION CADRE
17Ø   REM
18Ø   DATA  &B 1 1 1 1 1 1 1 1, &B 1 1 1 1 1 1 1 1
19Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
2ØØ   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
21Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
22Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
23Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
24Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
25Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
26Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
27Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
28Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
29Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
3ØØ   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
31Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
32Ø   DATA  &B 1 Ø Ø Ø Ø Ø Ø Ø, &B Ø Ø Ø Ø Ø Ø Ø 1
33Ø   DATA  &B 1 1 1 1 1 1 Ø Ø, &B Ø Ø 1 1 1 1 1 1
34Ø   S2$=« »:S3$=« »:REM LECTURE LUTIN N° 1
350   FOR   T=1 TO 16
36Ø   READ  A$, B$
37Ø   S2$=S2$+CHR$(VAL(A$))
38Ø   S3$=S3$+CHR$(VAL(B$))
39Ø   NEXT  T
```

```
400   REM
410   REM      INITIALISATION OBSTACLE
420   REM
430   DATA     1,128,0,0
440   DATA     1,128,96,6
450   DATA     1,128,112,14
460   DATA     1,128,56,28
470   DATA     1,128,28,56
480   DATA     1,128,14,112
490   DATA     1,128,7,224
500   DATA     255,255,3,192
510   DATA     255,255,3,192
520   DATA     1,128,7,224
530   DATA     1,128,14,112
540   DATA     1,128,28,56
550   DATA     1,128,56,28
560   DATA     1,128,112,14
570   DATA     1,128,96,6
580   DATA     1,128,0,0
590   Q1$=« »:Q2$=« »:Q3$=« »:Q4$=« »
600   FOR T=1 TO 16
610   READ V1, V2, V3, V4
620   Q1$=Q1$+CHR$(V1)
630   Q2$=Q2$+CHR$(V2)
640   Q3$=Q3$+CHR$(V3)
650   Q4$=Q4$+CHR$(V4)
660   NEXT T
670   REM
680   REM   24 POSITIONS OBSTACLES
690   DIM   PO(1,23)
700   DATA 8,32,64,8,160,16,160,168,80,112,32,128
710   DATA 176,128,48,64,224,112,168,232,32,80,104,144
720   DATA 112,32,136,136,216,40,56,160,200,8,128,64
730   DATA 168,56,120,104,176,88,96,64,16,168,232,80
735   DATA 112,168
740   FOR T=0 TO 23
750   READ XP,YP
760   PO(O,T)=XP:PO(1,T)=YP
770   NEXT T
780    REM
```

```
79Ø     REM GRAPHISME
8ØØ     REM
81Ø     SCREEN 2,2
82Ø     SPRITE$(1)=S$:REM CADRE
83Ø     SPRITE$(2)=S2$+S3$:REM BALLE
84Ø     SPRITE$(3)=Q1$+Q2$:REM OBSTACLE=+
85Ø     SPRITE$(4)=Q3$+Q4$:REM OBSTACLE=CROIX
86Ø     C=0:N=RND(-TIME)
87Ø     ON INTERVAL=3Ø GOSUB 138Ø
88Ø     INTERVAL ON:REM SEQUENCE DES POSITIONS
89Ø     ON SPRIT GOSUB 133Ø
9ØØ     PUT SPRITE Ø,(Ø,145),1Ø,2:REM CADRE
91Ø     X=2Ø:Y=Ø:REM POSITION DEPART
92Ø     PERDU=Ø
93Ø     I=2:REM INCREMENT DE DEPLACEMENT
94Ø     REM
95Ø     REM EVOLUTION DE LA BALLE
96Ø     REM
97Ø     SPRITE ON
98Ø     PUT SPRITE 1,(X,Y),13,1
99Ø     IF PERDU=1GOTO126Ø
1ØØØ    SPRITE STOP
1Ø1Ø    D=STICK(Ø):REM QUEL DIRECTION?
1Ø2Ø    IF D=Ø THEN D=M:REM D=DEPLACEMENT PRECEDENT
1Ø3Ø    IF D=1THEN Y=Y-I
1Ø4Ø    IF D =2THEN X=X+I:Y=Y-I
1Ø5Ø    IF D=3THEN X=X+I
1Ø6Ø    IF D=4THEN X=X+I:Y=Y+I
1Ø7Ø    IF D=5THEN Y=Y+I
1Ø8Ø    IF D=6THEN X=X-I:Y=Y+I
1Ø9Ø    IF D=7THEN X=X-I
11ØØ    IF D=8THEN X=X-I:Y=Y-I
111Ø    REM TEST SI SORTIE ECRAN
112Ø     IF X<Ø OR X>25Ø OR Y<Ø OR Y>1Ø5GOTO 126Ø
113Ø    REM TEST SI BALLE DANS CADRE
114Ø     IF X<222 AND X>212 AND Y<157 AND Y>147 GOTO 12ØØ
1150    M=D:REM MEMOIRE DEPLACEMENT
116Ø    GOTO 97Ø
117Ø    REM
118Ø    REM AFFICHAGE GAGNE
```

```
1190    REM
1200    INTERVAL OFF
1210    PRINT TAB(15)« GAGNE »
1220    GOTO 1220
1230    REM
1240    REM AFFICHAGE PERDU
1250    REM
1260    INTERVAL OFF
1270    PUT SPRITE 1,(X,209),13,1
1280    PRINT TAB(15) « PERDU »
1290    GOTO 1290
1300    REM
1310    REM INTERRUPTION SPRITE
1320    REM
1330    PERDU=1
1340    RETURN
1350    REM
1360    REM INTERRUPTION INTERVAL
1370    REM
1380    C=(C+1) MOD2
1390    Z=INT (RND(1)*24+2)
1400    PUT SPRITE Z, (PO(0,Z-2),PO(1,Z-2)),1,3+C
1410    RETURN
```

3.2 POSSIBILITES SONORES DU BASIX MSX

Après avoir décrit les nombreuses possibilités graphiques apportées par le Basic MSX, il est normal d'attendre les mêmes qualités au niveau des commandes sonores. Le circuit générateur de sons choisi par le standard permet de répondre à cette attente, et cela malgré l'existence de deux instructions seulement.

La première permet, par une syntaxe simple, de composer de véritables mélodies grâce à un macro-langage qui comporte de nombreuses commandes de définition et de contrôle des notes.

La deuxième est d'un usage plus spécialisé puisqu'elle permet de créer des sons dans une gamme de fréquence très large. De plus, il

est possible de générer un bruit et de le mixer avec des notes pour obtenir des effets sonores. Par conséquent cette commande, qui accède directement aux registres du circuit générateur de sons, est plus particulièrement utilisée pour accompagner des programmes de jeux.

Pour compléter cette introduction sur les possibilités sonores du Basic MSX, il ne reste qu'à indiquer que le générateur de sons travaille sur 3 voies en parallèle et presque complètement indépendantes les unes des autres. Ceci permet donc de générer des accords.

Mais passons à la description et à l'utilisation de ces deux instructions :

3.2.1 Génération de notes et de mélodies

Cette génération s'effectue par l'intermédiaire de l'instruction :

PLAY <chaîne de caractères pour la voie 1>
[,<chaîne de caractères pour la voie 2>]
[,<chaîne de caractères pour la voie 3>]

Cette commande permet de jouer des notes de la gamme en fonction de divers paramètres : volume, octave, modulation... Un macro-langage spécifique du Basic MSX intègre toutes ces caractéristiques. En effet, l'instruction PLAY a été conçue et est utilisée de la même façon que la commande DRAW : chaque chaîne de caractères est composée de commandes musicales d'un caractère auquel est associée une valeur numérique.

Tout d'abord, le programmeur doit indiquer les notes qui seront jouées par le générateur de sons. Il dispose de deux moyens :

— Soit spécifier les notes parmi la gamme et indiquer dans quel octave elles se jouent.

La gamme est codée avec les sept premières lettres de l'alphabet : A pour le la, B pour le si... Une option peut être rajoutée pour le dièse (# ou +) et pour le bémol (−).

Huit octaves peuvent être jouées par le générateur de son. L'octave des notes indiquées à la suite de la commande est programmée par la macro-instruction O<n>. Le paramètre n est

compris entre 1 et 8. Par défaut, n est égal à 4. Une octave commence par le do (lettre C).

— Soit indiquer la valeur de la note parmi les 96 existantes et réparties sur les 8 octaves. La syntaxe de la commande est N<n> avec n compris entre Ø et 96 : la valeur Ø correspond à un silence.

Pour la correspondance entre les deux sortes de déclaration, il suffit de savoir que la note C (ou do) de la quatrième octave est équivalente à la note N36. A partir de là, il faut savoir bien compter et ne pas oublier de notes.

Le programmeur ou compositeur dans cette partie du livre doit ensuite définir la durée des notes. Il dispose pour cette fonction de la macro-commande L<n>, n étant compris entre 1 et 64.

La table suivante donne une idée du codage de la durée des notes.

L1 correspond à une ronde
L2 à une blanche
L4 à une noire
L8 à une croche
...
L64 à une quadruple croche.

Les valeurs intermédiaires de n existent aussi (la valeur de n est par défaut 4, correspondant à une noire) .

Le programmeur peut indiquer la durée d'une note de deux manières dans la chaîne de commande :

— soit, par exemple L16A ;

— soit par le même exemple A 16.

Le macro-langage musical peut définir des silences entre les notes. La syntaxe de la commande est : R<n> avec n compris entre 1 et 64. Par défaut, n est aussi égal à 4.

Une autre commande de l'instruction PLAY permet d'augmenter la durée d'une note en la pointant. La commande est .. Placée derrière une note, elle multiplie sa durée par un et demi. Une note peut être pointée plusieurs fois.

Il ne reste plus au programmeur qu'à définir le tempo de la mélodie : dans le cas du Basic MSX, il s'agit de fixer le nombre de

noires exécutées en une minute. La commande est alors T<n> avec n compris entre 32 et 255. Par défaut, la valeur est 12Ø.

Il est possible de régler le volume de chaque voie par la commande V<n> (n compris entre Ø et 15 inclus). Par défaut, n est égal à 8. Pour cette instruction, chaque note a une amplitude sonore fixe durant toute la durée spécifiée.

Mais il est aussi possible de moduler le niveau sonore d'une note, il suffit de choisir une forme d'enveloppe parmi les 9 proposées et de fixer la période de modulation, dans le cas où celle-ci est périodique, ou la durée s'il s'agit d'une modulation limitée dans le temps (accroissement de volume, diminution...). Pour initialiser ces deux paramètres, il existe deux macro-commandes :

— S<n> avec n compris entre Ø et 15 qui définit la forme de l'enveloppe. Le tableau des enveloppes existantes est donné au paragraphe 6 de l'instruction SOUND. En effet l'instruction ne fait qu'initialiser, pour chaque note, le registre 13 du générateur de son (voir instruction SOUND).

— M<n> avec n compris entre Ø et 65535 (par défaut n=255). Cette commande fixe la période de la modulation. Pour plus de détails sur cette fonction, le paragraphe 5 de l'instruction SOUND explique et définit la correspondance entre la valeur <n> et la période.

Il est fort possible que des séquences de caractères se retrouvent à de nombreux endroits dans les instructions PLAY. Il est alors plus pratique d'assigner à ces séquences un nom de variable. Pour l'exécution des opérations indiquées, il existe une macro-commande : X<variable>. Elle permet d'exécuter la sous-chaîne de caractères assignée à la variable spécifiée.

Pour toutes ces macro-instructions, l'argument <n> peut être soit une constante entière, soit une variable entière. Dans ce dernier cas, la syntaxe est la suivante : <macro-commande>=<variable>.

Exemple

```
NB=2ØØ
FR=5ØØØ
```

```
PLAY «T=NB ;O3CDE»
PLAY «S1ØM=FR ;O4CDE»
```

Maintenant que sont définies toutes les macro-commandes existantes pour l'instruction PLAY, quelques remarques sont à faire à propos de l'ordre d'apparition de ces macro-commandes, dans la chaîne de caractères :

— généralement la valeur des notes (lettres A à G) se placent en fin de chaîne. En effet le programmeur doit définir le tempo, le volume, la modulation ou l'octave avant d'indiquer au générateur de son quelle note jouer ;

— par contre les commandes L (durée de note) et R (présence de silence) sont souvent intercalées entre les valeurs des notes.

Les remarques ne sont en aucune manière restrictives car elles iraient à l'encontre des possibilités du générateur de sons. Elles ne sont indiquées qu'à titre d'information.

Exemple

```
PLAY «O3CDE»
PLAY «T2ØØO3CDE
PLAY «T2ØØO3CL1DL2E»
PLAY «T2ØØO3CL1V5DL2V15E»
PLAY «O3CR1DR1ØE»
PLAY «S1ØM5ØØØO4CDE»
```

3.2.2 Génération de sons

Nous allons étudier maintenant l'instruction qui permet d'accéder directement aux registres du générateur de sons.

*** SOUND <registre du générateur de son>,<valeur à affecter>**

Cette instruction permet d'accéder directement aux différents registres de commande du générateur de son PSG *(Programmable Sound Generator)*. Ce circuit dispose de 13 registres :

— de RØ à R5 : programmation des tons sur les 3 voies.
— R6 : contrôle le générateur de bruit
— R7 : contrôle du mixage ton/bruit sur chaque voie.

— R8 à R1Ø : sélection de l'amplitude des notes (fixes ou enveloppes)
— R11 à R13 : programmation de la forme de l'enveloppe et de la période.

1) Programmation des tons

Pour chaque voie, il est possible de définir la fréquence de la note désirée par l'intermédiaire de 2 registres de 12 bits. A partir de cette fréquence, un petit calcul est nécessaire pour initialiser correctement les 2 valeurs :

$$\text{valeur désirée} = \frac{3579545}{16 \times \text{fréquence désirée}}$$

8 bits poids faibles : (valeur désirée) AND 255
4 bits poids forts : (valeur désirée) / 256

Pour la voie A, le registre bas est RØ, le registre haut est R1
Pour la voie B, le registre bas est R2, le registre haut est R3
Pour la voie C, le registre bas est R4, le registre haut est R5

Exemple d'utilisation

```
1Ø INPUT «ENTRER LA FREQUENCE :»;A
2Ø F = 3579545/(16*A)
3Ø H = F/256
4Ø L = F AND 255
5Ø SOUND Ø,L
6Ø SOUND 1,H
7Ø SOUND 8,15:REM VOLUME VOIE A
8Ø SOUND 7,254:REM VALIDATION VOIE A
9Ø GOTO 1Ø
```

2) Contrôle d'amplitude

Il existe deux possibilités pour l'amplitude d'un ton.

— soit fixe
— soit suivant une enveloppe prédéfinie.

Pour chaque voie, 5 bits sont à initialiser suivant le choix du programmeur :

— le bit 4 de chaque registre correspond à la sélection :
  Ø pour une amplitude fixe.
  1 pour une enveloppe prédéfinie.

— les bits Ø à 3 indiquent l'amplitude sonore codée en valeurs croissantes.

Le registre 8 contrôle l'amplitude de la voie A
Le registre 9 contrôle l'amplitude de la voie B
Le registre 1Ø contrôle l'amplitude de la voie C

Exemple de programmation

```
1Ø SOUND Ø,1ØØ
2Ø SOUND 1,Ø
3Ø SOUND 7,254:REM VALIDATION DE LA VOIE A
4Ø FOR I=15 TO Ø STEP - 1
5Ø SOUND 8,1
6Ø FOR J=1 TO 2ØØ:NEXT J:REM DELAI
7Ø NEXT I
```

3) Générateur de bruit

La période de bruit peut être fixée sur les 5 bits de poids faibles du registre 6.

4) Contrôle de mixage

Le générateur de son permet de mixer, pour une même voie, un bruit et/ou un ton. Ce contrôle s'effectue par l'intermédiaire du registre 7.

b_7	b_6	b_5	b_4	b_3	b_2	b_1	$b_Ø$
non utilisés		sélection du bruit			sélection du ton		
en Basic		C	B	A	C	B	A

bit i = Ø ; valide le ton ou le bruit dans la voie i.
bit i = 1 ; inhibe le ton ou le bruit dans la voie i.

5) Contrôle de la période de l'enveloppe.

En mettant à 1 le bit 4 du registre de contrôle d'amplitude de chaque voie, le programmeur module les tons suivant des enveloppes prédéfinies. Deux paramètres sont alors nécessaires : la forme et la période de la modulation. Les registres 11 et 12 du

générateur de son définissent la fréquence. A partir de la valeur désirée, un petit calcul est nécessaire pour initialiser les registres.

$$\text{valeur désirée} = \frac{3579545}{256 \times \text{fréquence désirée}}$$

Une fois codés en binaire, les 8 bits de poids faibles se placent dans le registre 11 et les 8 bits de poids forts dans le registre 12.

Exemple d'utilisation

```
1Ø    SOUND Ø,1ØØ
2Ø    SOUND 1,Ø:REM TON SUR VOIE A
3Ø    SOUND 7,&B1111111Ø:REM VALIDATION
4Ø    SOUND 8,16:REM AMPLITUDE = ENVELOPPE
5Ø    SOUND 13,14:REM CHOIX FORME
6Ø    S = Ø.5:REM 0.5 HERTZ
7Ø    L = 3579545/(256 * S) AND 255
8Ø    H = 3579545/(256*S*256)
9Ø    SOUND 11,L
1ØØ   SOUND 12,H
11Ø   END
```

6) Choix de la forme de l'enveloppe

Il s'effectue par le registre 13 parmi les 9 formes possibles. Le tableau suivant récapitule toutes les enveloppes :

Valeur à initialiser	Enveloppe
Ø, 1, 2, 3, 9	
4, 5, 6, 7, 15	
8	
1Ø	
11	
12	

13

14

Grâce à ces deux instructions très puissantes, le programmeur peut, s'il est un peu musicien, composer de petites mélodies. Par exemples, voici deux programmes qui génèrent deux morceaux très connus par les téléphiles.

```
1Ø    REM   HISTOIRES SANS PAROLES
2Ø    PLAY  «T12Ø»,«T12Ø»,«T12Ø»
3Ø    PLAY  «V1Ø»,«V1Ø»,«V1Ø»
4Ø    M1$   = «O6L16ECO5GE»
5Ø    M2$   = «O5CEO6AO5E»
6Ø    M3$   = «GL8O6AL16O5E»
7Ø    M4$   = «L5G»
8Ø    M5$   = «GO5CBD»
9Ø    M6$   = «L4C»
1ØØ   M7$   = «L8GL16F+GL8G+L4GL8BL4O6C»
11Ø   B1$   = «L8O3CO4E»
12Ø   B2$   = «L8O2GO4E»
13Ø   B3$   = «L8O2GO4B»
14Ø   B4$   = «O2L8GL16F+GL8G+L4GL8BL4O3C»
15Ø   D1$   = «L8O2CO6G»
16Ø   D2$   = «L8O1GO6G»
17Ø   D3$   = «L8O2CO4D»
18Ø   D4$   = «O1L8GL16F+GL8G+L4GL8BL4O2C»
19Ø   FOR  I = 1 TO 2
2ØØ   PLAY  M1$,B1$,D1$
21Ø   PLAY  M2$,B2$,D2$
22Ø   PLAY  M3$,B1$,D1$
23Ø   PLAY  M4$,B3$,D3$
24Ø   PLAY  M1$,B1$,D1$
25Ø   PLAY  M2$,B2$,D2$
26Ø   PLAY  M5$,B3$,D3$
27Ø   PLAY  M6$,B1$,D1$
28Ø   NEXT  I
29Ø   PLAY  M7$,B4$,D4$
```

Remarque

Le caractère O représente la lettre O et non le chiffre zéro

```
1Ø    REM  LES ANIMAUX DU MONDE
2Ø    FOR  I = 1 TO 2
3Ø    PLAY «T1ØØ», «T1ØØ», «T1ØØ»
4Ø    PLAY «V1Ø», «V1Ø», «V1Ø»
5Ø    M1$  = «O5L8GL16F +FL8EL16DC»
6Ø    M2$  = «O4L16BO5L32AR32AR32AR32L4A»
7Ø    M3$  = «O4L16BO5L32AR32AL16GL8FL16ED»
8Ø    M4$  = «O5L16AGF +AL4G»
9Ø    M5$  = «L2C»
1ØØ   B1$  = «L16O3CO4EO2GO4EO3CO4EO2GO4E»
11Ø   B2$  = «L16O2DO4FO2GO4FO3DO4FO2GO4F»
12Ø   B3$  = «L16O2DO4FO2GO4FO3DO4FO2GO4F»
13Ø   B4$  = «L16O2CO4EO2GO4EO3CO2GO2AB»
14Ø   D1$  = «L16O2CO4GO1GO4GO2CO4GO1GO4G»
15Ø   D2$  = «L16O1DO4BO1GO4BO2DO4BO1GO4B»
16Ø   D3$  = «L16O1DO4BO1GO4BO2DO4BO1GO4B»
17Ø   D4$  = «L16O1CO4GO1GO4GO2CO1GO1AB»
18Ø   PLAY M1$, B1$, D1$
19Ø   PLAY M2$, B2$, D2$
2ØØ   PLAY M2$, B2$, D2$
21Ø   PLAY M4$, B4$, D4$
22Ø   PLAY M1$, B1$, D1$
23Ø   PLAY M2$, B2$, D2$
24Ø   PLAY M3$, B3$, D3$
25Ø   PLAY M5$, B4$, D4$
26Ø   NEXT I
```

3.2.3 Fonction associée PLAY

La syntaxe de la fonction PLAY est PLAY(n), où n est le numéro de l'une des trois voies du générateur de sons (n = 1, 2 ou 3)

Le résultat de cette fonction est l'état de la voie testée :

– 1	voie en train d'émettre
0	voie au repos

PLAY (Ø) permet de tester simultanément l'état des trois voies.

CHAPITRE 4

Conclusion

La norme MSX connaît actuellement un grand succés au Japon. Les plus grandes firmes japonaises de micro-informatique s'y sont ralliées et plus d'une trentaine de machines sont déjà disponibles.

La percée européenne et américaine de MSX est plus timide. Les applications professionnelles font maintenant largement appel à des machines à base de microprocesseurs 16 ou 32 bits. Le marché des micro-ordinateurs 8 bits ne peut que stagner, voire décroître.

Les machines MSX seront certainement vouées aux applications familiales et possèdent des atouts non négligeables pour s'imposer sur ce marché très disputé : très grandes possibilités graphiques et sonores, nombreuses extensions, volumineuse bibliothéque de logiciels, mais aussi, et ce fut notre argument de départ, compatibilité de toutes les machines MSX entre elles.

Saluons l'initiative de Microsoft qui a créé le premier standard micro-informatique. Nous ne pouvons qu'espérer que les construc-

teurs trouvent avec la norme MSX un terrain d'entente pour aller de l'avant.

A propos, le MSX-SX *(MSX-Super eXtended),* c'est pour quand ?

ANNEXE 1

Jeu de caractères — caractères spéciaux

caractère	*action*
—	*blanc*
=	*signe égal ou symbole d'assignation*
+	*addition*
–	*soustraction*
*	*multiplication*
/	*« slash » ou division*
^	*exponentiation*
(	*parenthèse ouvrante*
)	*parenthèse fermante*
%	*pourcentage*
#	*dièse*
$	*dollar*
!	*point d'exclamation*
[	*crochet gauche*
]	*crochet droit*
,	*virgule*
.	*point décimal*
'	*apostrophe*
;	*point-virgule*
:	*deux points*
&	et *commercial*
?	*point d'interrogation*
<	*inférieur à*
>	*supérieur à*
	division entière
@	a *commercial*
-	*soulignage*
<rubout>	*détruit le dernier caractère frappé*
<escape>	*accès aux caractères de commande*
<tab>	*curseur à la prochaine zone d'affichage*
<line feed>	*curseur à la prochaine ligne physique*
<carriage return>	*termine et valide l'entrée de ligne*

ANNEXE 2

Codes et messages d'erreurs

Code	Message	*Description de l'erreur*

1 *Next without for* (NEXT sans FOR)

Une variable dans une instruction NEXT ne correspond pas à celle prévue. L'instruction FOR ne convient pas.

2 *Syntax Error* (erreur de syntaxe)

La ligne en cours d'exécution contient quelques séquences incorrectes de caractères (parenthèses mal placées, commande ou instruction mal orthographiée, ponctuation incorrecte, etc.)

3 *Return Without Gosub* (RETURN sans GOSUB)

Une instruction RETURN est rencontrée alors qu'il n'y a pas eu d'instruction d'appel à un sous-programme.

4 *Out of Data* (hors de données)

Une instruction READ est exécutée alors qu'il n'y a pas ou plus de données à lire.

5 *Illegal Function Call* (appel illégal de fonction)

Un paramètre qui est hors limite est passé dans une fonction mathématique ou dans une fonction chaîne. Cette erreur peut apparaître dans les cas suivants :

— une déclaration négative ou trop grande
— un argument nul ou négatif avec LOG
— un argument négatif avec SQR
— une mantisse avec un exposant non entier
— un CALL à une fonction USR pour laquelle l'adresse de début n'a pas encore été donnée
— un argument incorrect avec MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR... ON... GOTO

6 *Overflow* (Dépassement)

Le résultat d'un calcul est trop grand pour être représenté dans le format MSX Basic.

7 *Out of Memory* (hors mémoire)

Le programme est trop grand ou a trop de boucles avec FOR, ou trop de GOSUB, ou trop de variables, ou d'expressions trop compliquées.

8 *Undefined Line Number* (numéro de ligne non défini)
Une ligne inexistante est appelée dans une expression GOTO, GOSUB, IF... THEN... ELSE... ou DELETE.

9 *Subscript Out of Range* (déclaration hors limite)
Un élément d'un tableau est appelé soit avec une valeur qui est hors dimension du tableau, soit avec une fausse valeur.

1Ø *Redimensionned Array* (tableau redimensionné)
Deux instructions DIM sont données pour le même tableau.

11 *Division by Zéro* (division par zéro)
Une division par zéro est rencontrée, ou bien le résultat d'une opération se trouve élevé à une puissance négative.

12 *Illegal Direct* (instruction illégale en mode direct)
Une instruction qui n'est pas autorisée en mode direct est entrée comme une commande directe.

13 *Type Mismatch* (désaccord de type)
Un nom de variable chaîne est donnée à une valeur numérique ou vice versa. Un argument chaîne est donné à une fonction qui attend un argument numérique ou vice versa.

14 *Out of String Space* (hors de l'espace chaîne)
La place nécessitée par les variables alphanumériques a entraîné le Basic hors des limites de la mémoire restant libre. Le MSX Basic alloue l'espace chaîne de manière dynamique jusqu'à ce que la capacité maximum de mémoire soit atteinte.

15 *String Too long* (chaîne trop longue)
La longueur à une chaîne de caractère ne peut dépasser 255 caractères.

16 *String Formula Too Complex* (expression trop complexe)
Une expression est trop longue ou trop complexe. Elle doit être découpée en petites chaînes.

17 *Can't Continue* (ne peut continuer)
Un essai est fait pour tenter de continuer un programme qui :
— a été arrêté pour cause d'erreur
— a été modifié pendant l'exécution d'un BREAK
— n'existe pas

18 *Undefined User Function* (fonction utilisateur non définie)
Une fonction USR est appelée avant que la définition de la fonction ne soit donnée (instruction DEF).

19 *Device I/O Error* (erreur d'entrée-sortie)
Une erreur est apparue au niveau de la cassette, de l'imprimante, ou des disquettes.

20 *Verify Error* (erreur de vérification)
Le programme chargé est différent du programme sauvé sur cassette.

21 *No Resume* (pas de RESUME)
Une procédure de reconnaissance d'erreurs a été appelée mais elle ne contient pas l'instruction RESUME.

22 *Resume Without Error* (RESUME sans erreur)
Une instruction RESUME est rencontrée avant que la procédure de traitement d'erreur ne soit appelée.

23 *Unprintable Error* (erreur non imprimable)
Il n'y a pas de message d'erreur dans ce cas de figure. Ceci arrive habituellement dans le cas d'erreur sans code d'erreur défini.

24 *Missing Operand* (opérande manquant)
Une expression contient un opérateur sans opérande.

25 *Line Buffer Overflow* (dépassement de tampon de ligne)
La ligne entrée est trop longue.

ANNEXE 3

Sélection des couleurs

Code	Couleur
Ø	transparent
1	noir
2	vert
3	vert clair
4	bleu foncé
5	bleu clair
6	rouge foncé
7	cyan
8	rouge
9	rouge clair
1Ø	jaune foncé
11	jaune clair
12	vert foncé
13	magenta
14	gris
15	blanc

ANNEXE 4

Macro-commandes graphiques

U<n>	*Déplacement vers le haut*
D<n>	*Déplacement vers le bas*
L<n>	*Déplacement vers la gauche*
R<n>	*Déplacement vers la droite*
E<n>	*Déplacement en diagonale haut droite*
F<n>	*Déplacement en diagonale bas droite*
G<n>	*Déplacement en diagonale bas gauche*
H<n>	*Déplacement en diagonale haut gauche*
<n>	*Définit la longueur du déplacement qui sera multipliée par le facteur d'échelle*
S<n>	*Définit le facteur d'échelle*
B	*Déplace mais n'affiche pas*
N	*Déplace et retourne à la position initiale*
A<n>	*Définit l'angle n tel que :* *n = 0 correspond à 0 degré* *n = 1 " à 90 "* *n = 2 " à 180 "* *n = 3 " à 270 "*
C<n>	*Définit le numéro de couleur*
X<chaîne>	*Exécute la chaîne. Doit être terminé par un ;*

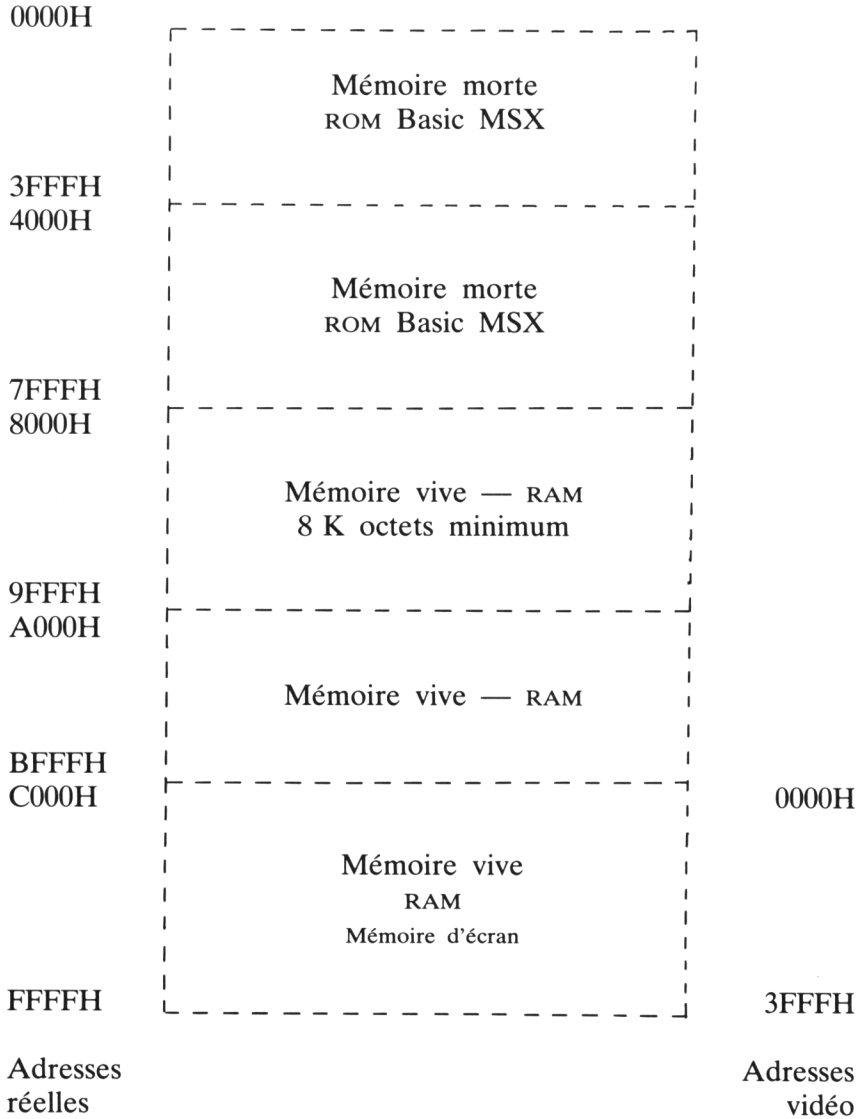
ANNEXE 5

Macro langage musique

A - G [#] [+] [−]	*Joue la note A « # » ou « + » signifie dièse et « − » signifie bémol.*
L<n>	*Définit la longueur de chaque note ; n peut être compris entre 1 et 64*
M<exp>	*Définit le timbre <exp> est la période définie par la commande S<n>. Cette commande n'est pas valide si la commande V est utilisée.*
N<n>	*Joue la note n 0≤n≤96 en 8 octaves . n = 0 signifie silence*
0<n>	*Définit l'octave de la note à jouer . n = 4 par défaut.*
R<n>	*Pause de n périodes 1≤n≤64 . même commande que L<n>*
S<n>	*Définit une forme d'enveloppe de bruit 0≤n≤15*
T<n>	*Tempo ou rythme 32≤n≤255 . n = 120 par défaut Définit le nombre n de L(4) dans une seconde*
V<n>	*V comme volume : définit la puissance sonore du son généré*
•	*Joue la note pendant une période égale à 3/2 de la période normale*
X	*Exécute les sous-chaînes*

ANNEXE 6

Organisation de la mémoire de base



ANNEXE 7

Spécifications matérielles minimales MSX

Unité centrale	: Z8Ø 3,579545 MHz
Mémoire morte	: 32 Ko
Mémoire vive	: 8 Ko (pouvant aller jusqu'à 64 Ko)
Mémoire vive vidéo	: 16 Ko
Contrôle vidéo	: TMS 9918 A ou TMS 9928A de Texas Instruments
Générateur de sons	: AY-3-8910, trois voies, huit octaves
Affichage écran	: texte : 24 lignes, 40 colonnes graphique : 256 × 192 pixels 16 couleurs
Clavier	: alphanumérique (QWERTY, AZERTY ou version japonaise) 73 touches dont 5 touches de fonction
Interface cassette	: format FSK, vitesse 1200 ou 2400 bauds
Interface disquette	: 3,5 pouces, 5,25 pouces, 8 pouces (format compatible MS-DOS)

Port pour cartouches de mémoire morte : 50 broches
Port standard pour deux manettes de jeux
Bus d'extension à 50 broches
Interface imprimante parallèle 8 bits
Interface vidéo RVB ou moniteur monochrome

INDEX

ABS 86
AND 39
ASC 87
ATN 86
AUTO 43

BASE 82
BEEP 53
BIN$ 83
BLOAD 94
BSAVE 93

CDBL 34
CHR$ 87
CINT 34
CIRCLE 109
CLEAR 53
CLOAD 94
CLOAD ? 94
CLOSE 79
CLS 106
COLOR 91
CONT 44
COS 86
CSAVE 94
CSNG 34
CSRLIN 90

DATA 56
DEF FN 49
DEF USR 50
DEFDBL 49
DEFINT 49
DEFSNG 49
DEFSTR 49
DELETE 44
DIM 48
DRAW 110

ELSE 70
END 53
EOF 80
EQU 39
ERASE 49
ERL 75
ERR 75
ERROR 76
EXP 87

FIX 83
FOR 66
FRE 84
FN 83

GOSUB 68
GOTO 67

HEX$ 84

IF 70
IMP 39
INKEY$ 59
INP 60
INPUT 57
INPUT# 81

INPUT$ 58
INSTR 88
INT 84
INTERVAL ON/OFF/STOP 73

KEY 98
KEY LIST 90
KEY ON/OFF 90
KEY ON/OFF/STOP 99

LEFT$ 88
LEN 88
LET 47
LINE 108
LINE INPUT 58
LINE INPUT# 80
LIST 44
LLIST 45
LOAD 93
LOCATE 90
LOG 87
LPOS 84
LPRINT 65
LPRINT USING 65

MAXFILES 79
MERGE 95
MID$ 88
MOD 36
MOTOR 92

NEW 45
NEXT 66
NOT 38

OCT$ 84
ON ERROR GOTO 75
ON... GOTO 69
ON... GOSUB 69
ON INTERVAL GOSUB 72
ON KEY GOSUB 98
ON SPRITE GOSUB 115
ON STOP GOSUB 73
ON STRIG GOSUB 95
OPEN 78
OR 39
OUT 65

PAD 97
PAINT 110
PDL 97
PEEK 51
PLAY 120
POINT 107
POKE 52
POS 90
PRESET 107
PRINT 60
PRINT USING 61
PRINT# 81
PRINT# USING 81
PSET 107
PUT SPRITE 113

READ 56
REM 54
RENUM 45
RESTORE 57
RESUME 75
RETURN 68
RIGHT$ 89
RND 84
RUN 46

SAVE 92
SCREEN 104
SGN 87
SIN 87
SOUND 123
SPACE$ 89

SPC 85
SPRITE ON/OFF/STOP 115
SPRITE$ 112
SQR 87
STICK 96
STOP 54
STOP ON/OFF/STOP 74
STRIG 96
STRIG ON/OFF/STOP 96
STRING$ 89
STR$ 89
SWAP 47

TAB 65
TAN 87
THEN 70
TIME 85
TRON 46
TROFF 46

USR 85

VAL 89
VARPTR 51
VDP 92
VPEEK 106
VPOKE 107

WAIT 55
WIDTH 91

XOR 39

&B 30
&H 29
&O 30

LA BIBLIOTHÈQUE EDIMICRO

- *Collection « Ordinateurs professionnels »*

APPLE

de Merly ..... **GUIDE DE L'APPLE**
Tome 1 : l'Apple standard
Tome 2 : les extensions
Tome 3 : les applications

MACINTOSH

Gaucherand ..... **MACINTOSH : outils, progiciels, applications**
Tenin, Van Thong ..... **Guide pratique du Sinclair QL**

- *Collection « Progiciels »*

APPLE

Bonnet, Dinh ..... **Multiplan sur Apple**

IBM PC

Bonnet, Dinh ..... **Lotus 1-2-3 à votre portée**
Bonnet, Dinh ..... **Multiplan sur IBM PC**

INTERET GENERAL

Bonnet, Dinh ..... **Mémento Multiplan**

- *Collection « Langages »*

Gaucherand, Lamoitier ..... **Fichiers en Basic par l'exemple.**
Perbost, Berthet ..... **Introduction à MSX**

- *Collection « Ouvrages de base »*

Lamoitier ..... **Le Traducteur Micro.**
Darnis, Van Thong ..... **Graphisme et CAO**

- *Collection « Ordinateurs familiaux »*

VG 5000

Amsler, Bardon ..... **Guide du VG 5000 Philips**
Amsler, Villemaud ..... **Jeux sur VG 5000 Philips**

MO5

Bieber, Perbost, Renucci ..... **Tout sur le MO5**
Perbost, Renucci ..... **Jeux sur MO5**

ATMOS/ORIC 1

Kosniowsky ..... **Nouveaux Jeux sur ATMOS**
Chane-Hune, Darbois ..... **Jeux Graphiques sur ATMOS**
Bayvejiel ..... **Guide de l'Oric**
Chane-Hune, Darbois ..... **Jeux sur Oric**
Viguier ..... **Premiers Pas en Programmation sur Oric**

SPECTRUM

Bridge, Carnell ..... **Aventures sur Spectrum**
Hurley ..... **Jeux Graphiques sur Spectrum**

COMMODORE 64

Fleurier, Meiller ..... **Jeux sur Commodore 64 : Jeux d'adresse et de hasard**
Fleurier, Meiller ..... **Jeux sur Commodore 64 : Jeux d'action et de réflexion**

ELECTRON

Bennani, Chaieb ..... **Graphisme et sons sur Electron**

PHILIPS C7420 VIDEOPAC +

Bardon, de Merly ..... **Jeux sur Philips C7420 Vidéopac +**

TO7

— ouvrages
Bieber, Perbost, Renucci ..... **Guide du TO7**
Perbost, Renucci ..... **Jeux sur TO7**

— logiciels sur cassette
Perbost, Renucci ..... **4 jeux avec manette pour TO7**
Perbost, Renucci ..... **6 jeux d'action et de réflexion pour TO7**

GENERAL

de Merly ..... **Ordinateur familial : Que choisir ?**

Nombreux autres titres à paraître. Catalogue sur simple demande.

EDIMICRO 121-127, avenue d'Italie, 75013 Paris

EXTRAITS DE PRESSE

GUIDE DE L'APPLE

« Un des meilleurs sur la place. »
Le Figaro.

« Le livre que nous attendions : complet, clair et pratique. »
Apple France.

« Toutes les étapes deviennent plus claires. »
Golden.

« Exposés clairs, à la portée de ceux qui ont des connaissances élémentaires en informatique. »
Minis et Micros.

MULTIPLAN SUR APPLE

« Nul doute qu'avec un tel outil vous gagnerez en temps et en efficacité. »
Microsoft France.

« Une bonne introduction... Un bon guide... Un bon investissement. »
Science et Vie Micro.

« Un livre stimulant... Un bon manuel utilisateur. »
Terminal.

FICHIERS EN BASIC PAR L'EXEMPLE

« Cet ouvrage est recommandé aux personnes amenées à entreprendre la programmation des fichiers du Basic, pour lesquels il constitue une étude pratique, sérieuse et en profondeur. »
Ordi Magazine.

MULTIPLAN SUR IBM PC

« Ce livre rendra de nombreux services. »
OPC.

« Des modèles de qualité professionnelle. »
L'Ordinateur Personnel.

JEUX GRAPHIQUES SUR SPECTRUM

« Ce livre constitue une véritable bibliothèque de jeux variés. »
Plein Jeu.

JEUX SUR COMMODORE 64

« Cet ouvrage permet aux débutants de s'initier au langage Basic en s'amusant... Un livre d'un bon niveau »
Tilt.

« Les programmes complets et pas trop longs utilisent bien les possibilités du C64. »
L'Ordinateur Individuel.

ORDINATEUR FAMILIAL :
QUE CHOISIR ?

« Ce livre a le mérite de simplifier le choix dans la jungle des boutiques informatiques. »
Décision Informatique.

PREMIERS PAS EN PROGRAMMATION

« Les exemples sont clairs et bien choisis, la méthode simple et rationnelle. »

MICR'ORIC, Revue des utilisateurs de l'ORIC.

« Une approche originale dans la littérature consacrée aux micros. »
Décision Informatique.

GUIDE DU TO7

« Ce guide a le privilège d'être présenté de façon claire, dans un style parfaitement accessible à tous. »
Micro 7.

« Un manuel de référence absolue »
Le Figaro.

JEUX SUR TO7

« De nombreux conseils et "trucs" pour une programmation rapide et efficace. »
Décision Informatique.

« Un excellent livre d'approche, plein de renseignements utiles. »
Micro 7.

GUIDE DE L'ORIC

« Tout ce qu'il faut savoir pour tirer le meilleur parti des possiblités — en particulier sonores et graphiques — de l'ORIC. »
Microsystèmes.

« Un best-seller. »
Livres Hebdo.

Nombreux autres titres à paraître. Catalogue sur simple demande.

EDIMICRO 121-127, avenue d'Italie, 75013 Paris

Achevé d'imprimer en mars 1985
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : mars 1985
Numéro d'imprimeur : 501031

INTRODUCTION A MSX

D. Berthet - A. Perbost

Le Basic standard MSX représente un bond en avant dans le domaine des langages pour micro-ordinateurs.

Il unifie et normalise les différentes versions du Basic qui, jusqu'à présent, changeaient constamment d'une machine à l'autre.

C'est déjà le langage des ordinateurs Sony, Hitachi, Canon, Toshiba, Sanyo, Yamaha et NEC. La plupart des constructeurs européens l'implanteront également sur leurs nouvelles machines.

« Introduction à MSX » vous donne tous les éléments pour maîtriser MSX et réaliser sans difficulté vos premiers programmes. Toutes les commandes et instructions sont expliquées, et illustrées par des exemples de programmes. Les instructions sonores et graphiques font l'objet d'un chapitre complet.

Chapitre 1 : Programmation et langage Basic.
Généralités — Démarrage — Mode calculateur et mode programme — Editeur pleine page — Constantes et variables — Opérateurs.

Chapitre 2 : Instructions et fonctions du Basic MSX
Instructions — Fonctions — Contrôle de l'écran

Chapitre 3 : Commandes graphiques et sonores

ISBN : 2-904457-29-1 108 F.

Edimicro

121-127, Avenue d'Italie 75013 Paris

Edimicro

INTRODUCTION A MSX

D. BERTHET
A. PERBOST